Universidad Carlos III de Madrid
Escuela Politécnica Superior

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

TÍTULO: **MIGRACIÓN A ANDROID DE UNA APLICACIÓN DE VISIÓN ARTIFICIAL PARA EL ÁMBITO EDUCATIVO**

Autora: **Patricia Uriol Resuela**

Tutores: **Iria Estévez Ayres** y **Jesús Arias Fisteus**

Octubre 2011

Nada hay bajo el sol que no tenga solución,
nunca una noche venció a un amanecer.

Agradecimientos

Muchas cosas son las que han pasado a lo largo de estos años de universidad. Ahora que es momento de hacer balance de todas ellas, me quedo solo con los aspectos positivos, los buenos momentos vividos, las alegrías recibidas y sobre todo, con la gente con la que lo he ido compartiendo.

Quiero comenzar agradeciendo a mi familia, toda la paciencia y apoyo que me han dado a lo largo de todos estos años. A mis padres Javier y Sagrario, que gracias a ellos me he convertido en la persona que soy ahora. A mi hermano Juanjo y a Marta por vuestros consejos y vuestra paciencia conmigo. A mis tíos y mi prima, que desde el comienzo me animaron y creyeron en mí. Emi, aunque he llegado tarde, espero que donde quiera que estés, puedas estar orgullosa de mí.

Gracias a Laura, mi amiga de siempre. Una amiga que me conoce casi mejor que yo misma, con la que he pasado muchas experiencias, tenido largas charlas y pasados buenos y malos momentos. Gracias por levantarme cuando me he caído, por escucharme cuando lo he necesitado y por estar siempre ahí.

Gracias también a mi gran amiga Vane. Siempre que tengo recuerdo de algún momento triste, ella ha estado ahí para sacarme adelante, gracias. Por supuesto agradecer también a Ale, la persona más sabia y que mejores consejos me ha dado. Gracias por enseñarme tanto y hacerme tener siempre los pies en el suelo.

Como no mencionar a Raquel, esa pequeña gran amiga que, a pesar de no llevar mucho tiempo conociéndonos, es como si llevásemos toda la vida juntas. Gracias por apoyarme siempre. Tú y Raúl tendréis siempre en mí una amiga. Muchas gracias a los dos por vuestra infinita paciencia en estos últimos tiempos.

Desde que comenzamos esta andadura por la universidad, Laura y Belén han estado ahí. Con ellas he compartido muchas experiencias dentro de las aulas pero muchas más fuera de ellas. Gracias a las dos por no dejarme decaer nunca, por estar siempre cerca y dejarme apoyarme en vosotras. No sé si hubiese sido capaz de conseguirlo sin vosotras, mil gracias.

A todas esas personas que han pasado por mi vida en todos estos años y que en mayor o menor media me han aportado valores, consejos y buenos momentos. A todos esos compañeros que se han convertido en amigos Nico, María y Alberto. A mis amigos de siempre María, Alberto , Irene y Juancar. Y a mis amigos y compañeros de Tid Almudena, Beatriz, David, Nuria y todos los demás. Gracias a todos ellos este proyecto se ha hecho realidad.

Muchas gracias a mis tutores Iria y Jesús, por su paciencia y tiempo dedicado. Gracias por estar siempre disponibles para echarme una mano y por brindarme la oportunidad de realizar este trabajo con vosotros.

Con este Proyecto Fin de Carrera se pone punto y final a una etapa de mi vida llena de nuevas vivencias, un gran esfuerzo realizado y muchas lecciones recibidas (tanto académicas como de la vida). A partir de ahora se abre un nuevo camino ante mi tan novedoso como lo fue éste al principio, y lo comienzo como en su día comencé la universidad, con ánimo, ganas e ilusión.

Resumen

En la actualidad, los dispositivos móviles se han convertido en una útil herramienta con la que el usuario disfruta de una amplia variedad de funcionalidades. Más allá de las aplicaciones básicas y llamadas telefónicas, estos pequeños terminales ofrecen una serie de funciones al usuario, tan complejas y provechosas, que los hacen indispensables en la sociedad. El uso extendido de estos aparatos ha provocado no solo la necesidad de crear nuevas aplicaciones para ellos, sino el hecho también de migrar y adaptar a dichos dispositivos programas ya existentes para ordenador.

Android es el sistema operativo de Google para dispositivos móviles. Es una plataforma que ofrece una infinidad de funcionalidades, un importante entorno de desarrollo y un innovador soporte tecnológico.

La visión artificial es un campo de la inteligencia artificial que está en auge en los últimos años, y cada día hay más estudios y desarrollos, investigando sus múltiples beneficios para la humanidad. Por medio del reconocimiento de patrones y procesado de imágenes (entre otras muchas técnicas), se consiguen objetivos como la detección, localización y reconocimiento de objetos en imágenes o la evaluación de resultados.

En este proyecto, se realizará una migración de una aplicación ya creada para ordenador, a un dispositivo móvil basado en la plataforma Android. El programa original se llama EyeGrade, está implementado en el lenguaje de programación Python, y consiste en la creación de un sistema de visión artificial orientado a un entorno educativo.

Concretamente, EyeGrade es un programa de ordenador creado para calificar un examen multirespuesta a través de una cámara *web*. Basado en la visión artificial, tiene como finalidad ayudar a los profesores a corregir exámenes a través de las nuevas tecnologías.

Palabras claves: Dispositivo Móvil, Android, Visión Artificial, Python

Abstract

Currently, mobile devices have become a useful tool with which the user enjoys a wide variety of features. Beyond the basic applications and phone calls, these small terminals offer a number of functions to the user, so complex and beneficial, that make them indispensable in society. The widespread use of these devices has caused not only the need to create new applications for them, but the fact also to migrate and adapt to these devices programs that already exist for computer.

Android is Google's operating system for mobile devices. It is a platform that offers a plenty of features, an important development environment and an innovative technological support.

Computer vision is a field of artificial intelligence that is on the rise in recent years, and there are more studies and development every day, researching their multiple benefits to humanity. Through pattern recognition and image processing (among many other techniques), are achieved goals such as the detection, localization and recognition of objects in images or the evaluation of results.

In this project, there will be a migration of an application already created for computer, to a mobile device based on the Android platform. The original program is called EyeGrade, it is implemented in the programming language Python, and is creating an artificial vision system aimed at an educational environment.

Specifically, EyeGrade is a computer program created to grade multiple choice question exams through a webcam. It is based on computer vision, with the aim of helping teachers to grade exams by new technologies.

Keywords: Mobile Devices, Android, Computer Vision, Python

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

AAPT (*Android Asset Packaging Tool*)

ADB (*Android Debug Bridge*)

ADT (*Android Development Tools*)

AIDL (*Android Interface Definition Language*)

API (*Application Programming Interface*)

ARM (*Advanced RISC Machine*)

ASE (*Android Scripting Environment*)

AVD (*Android Virtual Device*)

CLI (*Command Line Interface*)

CNRI (*Corporation for National Research Initiatives*)

CPU (Central Processing Unit)

CSV (*Comma-Separated Values*)

CWI (*Centrum Wiskunde & Informatica*)

DDMS (*Dalvik Debug Monitor Server*)

DX (*Dalvik Executable*)

E/S (Entrada-Salida)

FREJ (*Fuzzy Regular Expressions for Java*)

GPS (*Global Positioning System*)

GSM (*Global System for Mobile*)

GUI (*Graphical User Interface*)

HTML (*HyperText Markup Language*)

HVGA (*Half-size Video Graphics Array*)

IBM (*International Business Machines*)

IDE (*Integrated Development Environment*)

IPC (*Inter-Process Communication*)

JAR (*Java Archive*)

JDK (*Java Development Kit*)

JSON (*JavaScript Object Notation*)

LRU (*Least Recently Used*)

NDK (*Android Native Development*)

NTT (*Nippon Telephone and Telegraph*)

OCR (*Optical Character Recognition*)

OHA (*Open Handset Alliance*)

OI (*OpenIntents*)

OpenCV (*Open Source Computer Vision*)

P2P (*peer-to-peer*)

PC (*Personal Computer*)

PDA (*Personal Digital Assistant*)

PDF (*Portable Document Format*)

PNG (*Portable Network Graphics*)

PSF (*Python Software Foundation*)

Py4A (*Python for Android*)

RAM (*Random Access Memory*)

RGB (*Red, Green, Blue*)

RIM (*Research in Motion*)

RPC (*Remote Procedure Call*)

RTF (*Rich Text Format*)

SDK (*Software Development Kit*)

SL4A (*Scripting Layer for Android*)

SMS (*Short Message Service*)

SO (Sistema Operativo)

UI (*User Interface*)

VM (Virtual Machine)

Wi-Fi (*Wireless Fidelity*)

WYSIWYG (*What You See Is What You Get*)

XML (*Extensible Markup Language*)

Capítulo 1

Introducción

1.1 Introducción

Si hace unas décadas alguien hubiese afirmado que las tecnologías iban a avanzar tanto como para tener los dispositivos que existen en la actualidad, pocos le hubiesen creído. Tales avances han sido realizados en multitud de áreas, pero sin embargo, uno de los campos donde han tenido lugar los cambios más notorios ha sido el de las telecomunicaciones. Dentro de esta amplia área temática, dos de sus principales logros han sido Internet y la telefonía móvil.

Ambas creaciones son relativamente recientes, no teniendo más de cincuenta años, aunque su desarrollo ha sido y sigue siendo, ciertamente veloz e implacable. A diferencia de otros inventos pasados, los avances tecnológicos de los que estamos aquí tratando parecen no tener fin. El origen de Internet tiene lugar en la década de los 60, desarrollándose originalmente en el seno de las universidades y de los laboratorios americanos que trabajaban sobre proyectos militares [1], mientras que la telefonía móvil tuvo su punto de partida en Tokio en el año 1979 con la primera red celular de telefonía móvil pública instalada por la operadora NTT (*Nippon Telephone and Telegraph*) [2]. Ambas tecnologías se desarrollaron fuertemente a lo largo de los años, hasta que las dos convergieron en el mismo punto para lograr unirse y, a lo largo de un periodo largo y fluctuoso, llegar hasta la actualidad, donde a pesar de no captar todo el mercado en exclusividad, la mayoría de los teléfonos móviles ofrecen Internet, y muchos de los ciudadanos contemporáneos relacionan intrínsecamente Internet y móvil.

Las universidades han sido siempre un gran motor de avance de la sociedad, y con sus pequeñas o grandes contribuciones han ayudado a llegar hasta la situación actual. En las universidades surgen una gran cantidad de aplicaciones que inicialmente están orientadas al ámbito educativo, pero que posteriormente se extienden a otros campos de la vida.

De la unión entre las nuevas tecnologías y el siempre importante pilar de la universidad, surge este proyecto que aquí comienza. Un estudio que se basa en la visión artificial llevada a un entorno móvil y aplicado en el ámbito educativo.

1.2 Motivación y objetivos

Son muchos y muy distintos los campos que se han abarcado a lo largo de esta carrera y por consiguiente, también las opciones que se tienen para elegir. El hecho de que estemos inmersos en una autentica revolución tecnológica, ha sido uno de los motivos para seleccionar la ingeniería telemática y dentro de ella, el desarrollo de la tecnología móvil.

Según se desarrollaban los teléfonos móviles, éstos han ido disminuyendo en tamaño a la vez que aumentaban en funcionalidad. Sus funciones más básicas y primarias como eran la realización de llamadas o el envío y recepción de SMS (*Short Message Service*) han quedado ya como una mera característica del pasado, siendo en la actualidad simples aspectos que todos los componentes tienen.

Se ha pasado de competir en la calidad de la llamada, a que lo relevante ahora sean las distintas aplicaciones y su grado de aceptación por parte de los usuarios. Entre medias quedó la época en la que importaba la calidad de las imágenes al realizar fotografías o vídeos, así como el tamaño y estética del aparato. Hoy en día, estos aspectos suelen ser más parejos, siendo en la gran mayoría de los casos, terminales de pequeño tamaño, formas bonitas y con muy alta calidad de imagen. Por tanto, y como ya se ha mencionado, lo que se busca implementar ahora y destacar por ello, es conseguir dispositivos móviles con aplicaciones útiles para los usuarios, ya sean de entretenimiento y ocio, de obtención de información ó de cualquier otra temática, lo importante es que sean atractivas para los consumidores.

El principal aspecto característico de cada dispositivo móvil es el sistema operativo que utiliza. Existen diversos SOs (Sistemas Operativos) como Windows Mobile ó el iSO de Apple, sin embargo el sistema operativo elegido para desarrollar nuestra aplicación será Android, creado e implementado por la compañía Google. Dicha plataforma será estudiada y presentada con detalle a lo largo del presente trabajo.

Buscando obtener una aplicación que fuese de utilidad para los usuarios y estando en el entorno universitario, se ha procedido a trabajar sobre una aplicación dirigida a los profesores, y que ayuda a éstos, a crear y a corregir exámenes tipo test mediante la cámara del dispositivo móvil Android.

La aplicación en sí ya está creada y operativa para funcionar sobre un ordenador de sobremesa. El programa está escrito en lenguaje Python y hace uso de una cámara *web* conectada al ordenador. En este proyecto se llevará a cabo la migración de dicha aplicación a un entorno móvil, en concreto a un dispositivo basado en tecnología Android.

Esta aplicación ha ofrecido así mismo la oportunidad de juntar dos lenguajes de programación, Python y Java (lenguaje sobre el que se desarrolla Android), hecho que a su vez ha constituido la mayor complejidad del proyecto.

El desafío de conseguir que el programa principal, escrito en Python, funcionase para una aplicación móvil Android, con todo lo que esto conlleva, ha sido un gran punto de motivación a la hora de realizar este proyecto.

El objetivo principal por tanto de este proyecto, es migrar una aplicación, escrita en lenguaje Python, al sistema operativo para dispositivos móviles de Google (Android). Dicha aplicación tiene como funcionalidad característica, el ser capaz de captar con la cámara las soluciones escritas en un examen multirespuesta en función de una configuración inicial, facilitando posteriormente la información del análisis realizado del examen, información como la puntuación total o los fallos y aciertos conseguidos.

Para lograr dicho objetivo final, se tienen que ir consiguiendo antes una serie de pasos no menos importantes. Las fases que se desarrollarán a lo largo de este trabajo y que conforman los objetivos de este proyecto, son las siguientes:

Realizar un estudio de la plataforma Android: estudio general del sistema operativo de Google. Investigación de las principales características de la plataforma Android y de sus elementos más significativos.

Realizar un estudio de los otros componentes importantes que se van a utilizar: documentación sobre herramientas y elementos básicos que se emplean en la aplicación o en su adaptación a la nueva plataforma. Estudio del lenguaje de programación Python sobre el que está escrito el programa inicial, investigación sobre la herramienta que permite adaptar este lenguaje al SO Android, la herramienta *Scripting Layer for Android*.

Presentación del programa origen: introducción de la aplicación inicial de ordenador. Explicación de sus principales características y funcionalidades.

Análisis del diseño que se va a desarrollar: estudio del diseño a implementar y presentación de las alternativas de diseño. Enumeración de las funciones del programa, estudio de la arquitectura de la aplicación final Android y revisión de sus distintos casos de uso.

Crear un proyecto Android implementado en código Python: investigación y búsqueda de la manera de hacer posible la creación de una aplicación Android basada en el lenguaje Python.

Migrar el código original teniendo en cuenta todas sus necesidades: revisar los requisitos y necesidades que vayan surgiendo a lo largo de la migración de la plataforma. Realización de la aplicación adaptada a partir del código ya existente.

Crear una nueva forma de introducir los parámetros de entrada: adaptación de la aplicación con respecto a la comunicación inicial con el usuario. Modificación de la interacción original basada en la pantalla de línea de comandos.

Crear una nueva interfaz gráfica de usuario específica para Android: modificación en la forma de realizar la interfaz gráfica de usuario. Adaptación de ésta al nuevo entorno móvil de los dispositivos Android.

Gestionar la forma de tomar y visualizar imágenes con Android: creación de una nueva forma de realizar la captura de imágenes y su posterior visualización. Adaptación de estos procesos a las características que rigen la plataforma Android.

1.3 Estructura del documento

Este proyecto está dividido en siete capítulos y dos anexos finales, explicados todos ellos a continuación:

El primer capítulo, 'Introducción', presenta las motivaciones y objetivos del proyecto así como la estructura que se va a seguir en él.

En el segundo capítulo ('Estado del arte') se realiza una exposición general sobre algunos de los aspectos técnicos relacionados con este proyecto. Se comienza con un estudio detallado de la plataforma Android para dispositivos móviles en el que se describen por ejemplo, sus características generales, arquitectura o componentes básicos. Posteriormente, se describen también el lenguaje de programación Python, la herramienta *Scripting Layer for Android*, y la aplicación origen EyeGrade.

El tercer capítulo 'Alternativas de diseño' es una exposición del diseño inicial del sistema a realizar y la alternativa a éste, el diseño final implementado.

El cuarto capítulo 'Diseño del sistema', expone los requisitos necesarios del sistema a implementar, su arquitectura y los diferentes casos de uso que se tendrán. Presentando también los diagramas de flujo y de clases del sistema completo.

El quinto capítulo 'Implementación', ofrece detalladamente las decisiones de implementación que se han realizado.

El sexto capítulo 'Pruebas', explica las pruebas realizadas en el sistema, presentando las valoraciones y resultados obtenidos de ellas.

El séptimo capítulo, 'Conclusiones finales y trabajo futuro', declara los resultados obtenidos y objetivos conseguidos tras la finalización del sistema, trazando posteriormente posibles líneas futuras de desarrollo.

Por último, se encuentra la bibliografía y los dos siguientes anexos, 'Presupuesto' y 'Manual de instalación'.

Capítulo 2

Estado del arte

2.1 Introducción

Este capítulo es una presentación de las técnicas y herramientas que se van a utilizar a lo largo de este proyecto fin de carrera. Una introducción técnica de los recursos que van a intervenir en la migración de la aplicación EyeGrade desde el ordenador al dispositivo móvil Android.

Cinco son los puntos básicos que se tratarán en este capítulo: sistemas operativos móviles, plataforma Android, lenguaje de programación Python, herramienta *Scripting Layer for Android* y la aplicación inicial EyeGrade.

Se comenzará con una sección dedicada a presentar los sistemas operativos móviles. Definiendo algunos términos importantes y presentando su arquitectura y su situación de mercado actual.

En segundo lugar se realiza un estudio de la plataforma Android. Indicando, por ejemplo, sus características más importantes, su arquitectura, o los componentes de construcción de una aplicación.

El lenguaje de programación Python es el siguiente punto de estudio de este capítulo. Estudio que se realizará mediante la descripción de las características y elementos de dicho lenguaje.

Posteriormente se describe la herramienta *Scripting Layer for Android.* El instrumento a través del cual se consigue la unión del lenguaje de programación Python y la plataforma Android.

El último punto de este capítulo es el dedicado a la aplicación inicial EyeGrade para ordenador. Realizando una descripción del programa, indicando su funcionalidad y explicando sus principales características.

2.2 Sistemas Operativos Móviles

Un sistema operativo, según la Real Academia de la Lengua, es un programa o conjunto de programas que efectúan la gestión de los procesos básicos de un sistema informático, y permite la normal ejecución del resto de las operaciones [3]. Otra definición de los sistemas operativos es: 'administrador de recursos' [4], es decir, un SO es el responsable de administrar los recursos de un sistema informático de manera eficiente y equitativa. Los recursos del sistema incluyen los procesadores (CPUs*, Central Processing Units*), memorias (tanto física como virtual), los posibles dispositivos (discos), archivos y directorios, y conexiones de red (puertos). Por eficiente, se quiere decir que el sistema operativo debe tener como objetivo maximizar la utilización de recursos siempre que sea posible; y por equitativa, se refiere a que a los programas de los usuarios se les debe conceder una asignación ecuánime de los recursos durante su ejecución [4].

Tomando como punto de partida los primeros teléfonos móviles, los sistemas operativos utilizados por aquel entonces no constituían una característica que hiciera destacar un móvil sobre cualquier otro. La primera generación de teléfonos móviles, conocida como 1G, eran de gran tamaño y peso, y se utilizaba exclusivamente para llamadas de voz, lo que no quita para que a partir de los años noventa su demanda se fuese disparando año tras año. La siguiente generación (2G) que aparece en el mercado, empleaba el sistema GSM (*Global System for Mobile*) mejorando la calidad de voz en las llamadas y proporcionando alguna funcionalidad extra, como la mensajería de texto SMS o la identificación de llamadas. Desde ese momento y hasta llegar al punto actual, no ha pasado un gran periodo de tiempo, pero sí un avance tecnológico importante, en el que se ha ido desarrollando generación tras generación hasta llegar a la cuarta, también conocida como 4G.

En dicho proceso de evolución se pasó por la inclusión poco a poco de Internet en los móviles, haciendo necesario aumentar la velocidad de transmisión de datos. Este y otros muchos desarrollos tecnológicos, junto con la creciente demanda del mercado de nuevas y, cada vez, más sofisticadas prestaciones, hicieron posible la aparición de los hoy llamados *smartphones* o "teléfonos inteligentes". Dispositivos móviles de pequeño tamaño que además de funcionar como teléfonos permiten muchas más opciones de usabilidad, acercándose notoriamente a los ordenadores personales. Es decir, los *smartphones* son dispositivos multifunción con teléfono incluido, pero que cuentan además con cámara, conexión a Internet, reproductores multimedia, Wi-Fi (*Wireless Fidelity*) o servicios de GPS (*Global Positioning System*) entre otras muchas posibles aplicaciones.

Como se ha mencionado anteriormente, surgió un nuevo concepto, los dispositivos móviles, un nuevo grupo que fue necesario catalogar debido a que comenzaron a surgir dispositivos electrónicos no relacionados directamente con la telefonía pero si de tamaños similares y con algunas funcionalidades comunes, que llegaron a ser tan populares como los citados *smartphones*. Algunos ejemplos de estos aparatos son las cámaras digitales, reproductores de música, ordenadores portátiles ó PDAs (*Personal Digital Assistants*) entre otros. De todo lo expuesto se puede obtener la definición del término dispositivo móvil como, una familia heterogénea de aparatos electrónicos de tamaño reducido que permiten su movilidad, es decir, de dimensiones suficientemente pequeñas que hacen posible

transportarlos a la vez que se usan. Poseen conexión a Internet de forma permanente o intermitentemente, y una función principal, aunque suelen poder desarrollar otras funciones diferentes a la especifica del dispositivo. Estos dispositivos incorporan un sistema operativo y capacidades para el almacenamiento, computación y entrada-salida de datos (E/S). El reducido tamaño conlleva limitaciones en el *hardware*, lo que se ve reflejado en las restricciones que tienen en el consumo de energía, tamaño de la pantalla, memoria RAM (*Random Access Memory*) o capacidad del procesador.

En el caso de los ordenadores portátiles, los sistemas operativos empleados no se diferencian de los usados en los ordenadores de sobremesa debido a sus avanzadas características de memoria y procesamiento, llegando en algunos casos a poder ser superiores. Así pues, dichos dispositivos móviles emplearían los habituales sistemas operativos, sin embargo, existen otros que necesitan sistemas operativos fiables, de gran estabilidad y que se amolden a sus restrictivas capacidades (tanto de procesamiento de datos como de memoria) y es por ello que se crearon los sistemas operativos móviles.

Los sistemas operativos móviles son los sistemas operativos que controlan los dispositivos móviles. Sin embargo, no se puede decir que sea una simple adaptación de los SO usuales como Windows, Mac OS o Linux. Un sistema operativo móvil necesita una nueva arquitectura y diferentes características a fin de proporcionar servicios adecuados para los dispositivos de mano. Cada uno de los SO móviles emplea una arquitectura e implementación diferente. La Figura 2. 1 muestra una estructura general de los sistema operativo móviles, representada como una pila de seis capas: (1) aplicaciones, (2) interfaz gráfica de usuario, GUI (*Graphical User Interface*), (3) marco de la interfaz de programación de aplicaciones , API (*Application Programming Interface*), (4) multimedia, infraestructura de comunicación, y seguridad, (5) kernel del equipo, gestión de energía, y el kernel en tiempo real y (6) controlador de hardware [5].

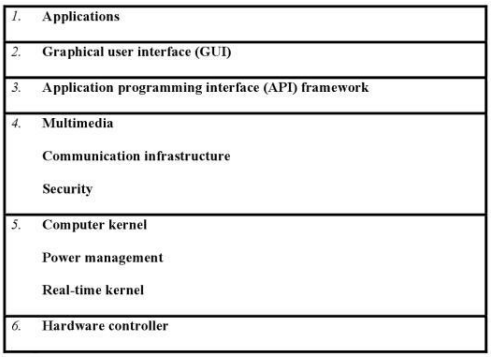


Figura 2. 1: Estructura general de un sistema operativo móvil [5]

Los sistemas operativos móviles más importantes y asentados en el mercado actual son los siguientes: Symbian OS, iSO de Apple, Windows Phone, BlackBerry de RIM (*Research In Motion*) y Android. En la actualidad el mercado está dominado por Symbian y Android, seguidos de BlackBerry e iSO. En la Figura 2. 2 se pueden observar los datos reportados por Gartner Inc con los valores de los años 2009-2010.

Worldwide Smartphone Sales to End Users by Operating System in 2010 (Thousands of Units)

Company	2010 Units	2010 Market Share (%)	2009 Units	2009 Market Share (%)
Symbian	111,576.7	37.6	80,878.3	46.9
Android	67,224.5	22.7	6,798.4	3.9
Research In Motion	47,451.6	16.0	34,346.6	19.9
iOS	46,598.3	15.7	24,889.7	14.4
Microsoft	12,378.2	4.2	15,031.0	8.7
Other Oss	11417.4	3.8	10432.1	6.1
Total	**296,646.6**	**100.0**	**172,376.1**	**100.0**

Source: Gartner (February 2011)

Figura 2. 2: Ventas de *smartphones* según SO en 2010 [6]

De la Figura 2. 2 se pueden sacar varias conclusiones, como por ejemplo, que el SO para *smartphones* que ha explotado de 2009 al 2010 ha sido Android, que cómo se puede apreciar, ha multiplicado por 10 las ventas al usuario final. Symbian, que sigue siendo el SO más empleado, ha aumentado en un 37,95% mientras que BlackBerry lo ha hecho en un 38.15%. En el caso de Windows Phone, se presupone que la reducción de sus ventas, que ha tenido lugar de 2009 a 2010, cambie con respecto a este año, con la novedosa alianza estratégica entre Microsoft y Nokia, que supuestamente hará que se recupere.

Symbian es un sistema operativo y plataforma de software diseñada para *smartphones* y que se encuentra actualmente mantenida por Nokia. Sus orígenes se encuentran en la alianza que tuvo lugar entre varias empresas de telefonía móvil siendo Nokia, Ericsson, Psion o Motorola algunas de ellas. El primer equipo que se implementó con Symbian, en el año 2000, fue el 'Nokia 9210 Communicator' [7]. RIM BlackBerry OS se centra en facilitar un buen funcionamiento y fue diseñado originalmente para los negocios. En 1996 se creó la primera BlackBerry, '900 Interactive Pager'. Recientemente se ha detectado un aumento en las solicitudes de aplicaciones fuera del ámbito empresarial, lo que ha conllevado a una considerable mejora, a la hora de ofrecer un soporte multimedia completo [8]. iSO (también conocido como iPhone OS) de Apple, es el SO de la compañía Apple Inc. y solo existe en productos de dicha compañía, ya que no vende la licencia para poder instalarse en ningún *hardware* de terceros. En el año 2007 se lanzó al mercado la primera versión del 'iPhone' [9]. El 15 de febrero del 2010 se presentó Windows Phone 7 [10], una nueva generación de SO móvil de Microsoft que sin ser la misma plataforma, es una evolución del anterior SO de Microsoft, Windows Mobile. Los primeros ejemplares con SO de Microsoft surgieron en 2000 y 2010, con Windows Mobile y Windows Phone 7 respectivamente [9]. Existen también algunos otros sistemas operativos en el mercado actual de menor importancia, ya que llevan poco tiempo, estos SOs son por ejemplo, Wophone, LiMO, Bada, WebOs ó Meego [11].

Android es el SO de código libre creado por la compañía Google para dispositivos móviles, sacando al mercado su primer terminal en el año 2008, el 'HTC Dream'. Como plataforma de desarrollo, Android es potente e intuitiva, permitiendo a los desarrolladores que nunca han programado para los dispositivos móviles, crear útiles aplicaciones de forma rápida y sencilla [12].

2.3 Plataforma Android

A continuación se va a proceder a presentar los principales conceptos relacionados con el SO Android. Dicha sección es necesaria a la hora de implementar tanto la aplicación que en este trabajo se va a desarrollar como cualquier otra, ya que antes de poder programar nada, se ha de asimilar correctamente los conceptos, y eso es lo que se pretende en esta unidad, aprender las nociones básicas de Android y entender cómo estas se relacionan entre sí.

2.3.1 Introducción

La plataforma Android se caracteriza principalmente por el hecho de ser código libre, lo que posibilita que todo aquel que posea conocimientos para ello pueda cambiar y rehacer a su medida el código de las aplicaciones. Esto hace posible que haya miles de personas en el mundo haciendo programas y no sólo una empresa, lo que a su vez consigue que Android sea un SO muy completo y con miles de aplicaciones siendo la mayoría gratis o muy baratas y encontrándose todas ellas en un mercado virtual llamado “market”.

Seguidamente se va a presentar la historia de Android, así como las diferentes versiones por las que se han ido pasando desde su creación. Las principales características de este SO, entre las que destaca la máquina virtual que posee (Dalvik) y por último los componentes principales, es decir, su arquitectura.

2.3.1.1 Historia

El origen de todo se sitúa en el año 2007 con la creación de OHA (*Open Handset Alliance*). OHA es un grupo de más de 80 empresas tecnológicas en las que están incluidos fabricantes de *hardware*, distribuidores de dispositivos móviles y desarrolladores de *software*. Entre las compañías más importantes de este consorcio cabe destacar las empresas de telecomunicaciones Motorola, HTC, T-Mobile y Qualcomm. La OHA representa, según sus palabras, un compromiso de conseguir de forma unida la aceleración de la innovación en los móviles y ofrecer a los consumidores experiencias móviles más enriquecedoras, mejores y menos costosas [13].

El 5 de Noviembre del 2007 concretamente, tiene lugar la presentación por parte de Google junto con *Open Handset Alliance* de Android, una compleja y a la vez, completa plataforma *software* para dispositivos móviles: un sistema operativo, *middleware* y las principales aplicaciones del dispositivo móvil. Android se orientó principalmente hacia los desarrolladores de código puesto que tanto Google como OHA apostaron desde sus comienzos por facilitar a éstos, los desarrolladores ajenos, las herramientas necesarias para escribir sus propios programas y así conseguir de la mejor manera posible, la realimentación gracias al *software* libre, el camino más fácil para implementar mejores aplicaciones *software* para los clientes.

La OHA esperaba proporcionar mejoras continuas en el *software* para los clientes por medio de la plataforma que proporciona y que es necesaria para poder desarrollar innovaciones para el móvil a más velocidad y de mayor calidad, sin cuotas de licencias para los desarrolladores de *software* ni para los fabricantes de dispositivos móviles. En última instancia, el éxito de Android como una plataforma móvil dependerá en gran medida del éxito de los asociados de OHA, que son los encargados de la liberación de terminales convenientes y servicios móviles que fomenten la adopción generalizada de los teléfonos Android. Mientras que los desarrolladores, tienen la oportunidad de crear nuevas aplicaciones móviles para Android, para así alentar a más empresas de tecnología móvil a formar parte de OHA [12].

Más que un sistema operativo móvil creado para una única implementación de *hardware*, Android fue diseñado para soportar una gran variedad de plataformas *hardware*, desde teléfonos con pantalla táctil a dispositivos sin ninguna pantalla. Más allá de eso, la no existencia de tasas de licencia ni *software* propietario, hacen que el coste para los fabricantes de dispositivos móviles al proporcionar variaciones de sus terminales móviles compatibles con Android sea relativamente bajo. Se esperaba que una vez que la demanda de *hardware* capaz de ejecutar aplicaciones de Android alcanzase un nivel suficientemente alto, aumentarían los fabricantes de dispositivos que se pusieran a producir cada vez más *hardware* a medida para satisfacer dicha demanda [12].

El primer teléfono que se pudo comprar con un SO Android fue el '*T-Mobile G1*' de la compañía telefónica T-Mobile, en Octubre del 2008 en Estados Unidos y un mes después (Noviembre del 2008) en Reino Unido. En España los primeros terminales con Android fueron el '*HTC Dream*' (T-Mobile G1) por medio de Movistar y el '*HTC Magic*' (T-Mobile G2) por parte de Vodafone, ambos en el año 2009. En la Figura 2. 3 se pueden apreciar ambos terminales.

(a) *HTC Dream* (b) *HTC Magic*

Figura 2. 3: Primeros terminales Android en España

2.3.1.2 Versiones

Desde sus inicios han sido varias las versiones que se han ido desarrollando y mejorando para el uso de dispositivos con Android. Desde que tuvo lugar la primera versión 1.0, utilizada en el primer dispositivo Android T-Mobile G1 (Octubre del 2008), hasta llegar a la actual 3.0, han sido diversas las actualizaciones del sistema operativo que se han

implementado, dando lugar a las diferentes versiones, cada una bajo un nombre distinto y coexistiendo siempre varias de ellas.

Así pues se va a proceder a presentarlas brevemente para tener una visión de cómo ha ido evolucionando esta plataforma.

A pesar de que este SO móvil no lleva mucho tiempo en el mercado (Android tiene sus orígenes en Octubre del 2008), la plataforma ha evolucionado en este tiempo, consiguiendo cada vez más soportes nuevos, tanto *hardware* como *software*. Con las diferentes versiones de la plataforma, se pueden hacer tres grandes agrupaciones según la importancia en la evolución de Android. Según esta diferenciación que se va a presentar aquí, se tendría el primer grupo con las dos primeras versiones, la 1.0 y la 1.1. Un segundo conjunto en el que se situarían las versiones principales ó más importantes hasta la fecha (versiones 1.5, 1.6, 2.1 y 2.2), y un tercer grupo en el que se encontrarían las últimas versiones, que ya se han comenzado a comercializar o están en ello, pero que aún no se han expandido demasiado por estar en proceso de implementación y desarrollo, o bien por ser especificas para ciertos dispositivos, como es el caso de HONEYCOMB para tabletas. En esta última agrupación de actualizaciones se encontrarían las versiones 2.3 ó GINGERBREAD y la versión ya mencionada 3.0 HONEYCOMB, de Diciembre del 2010 y de Enero del 2011 respectivamente. Será el segundo conjunto el que se explique un poco más detenidamente en esta sección debido a su gran relevancia en el mercado actual.

El primer conjunto de versiones está formado por la versión original 1.0 y la primera actualización 1.1, lanzadas respectivamente en Octubre del 2008 y Febrero del 2009. Ambas implementan los dos primeros niveles de la API de Android[1] (a la versión 1.0 de Android le corresponde "*API Level 1*" mientras que la 1.1 está asignada a la "*API Level 2*"). No hay grandes diferencias entre ambas, puesto que la 1.1 es una actualización que corregía algunos *bugs* y no traía cambios importantes.

La segunda agrupación está formada por las versiones 1.5 (CUPCAKE), 1.6 (DONUT), 2.1 (ECLAIR) y 2.2 (FROYO). Es en este grupo donde empiezan a surgir mejoras considerables, así como grandes novedades y avances con respecto a las versiones anteriores.

La primera, versión 1.5 ó también llamada CUPCAKE, se lanzó en Abril del 2009 y posee 4 actualizaciones (revisiones) dentro de ella misma, teniendo lugar la última de dichas *releases* en Mayo del 2010. Android 1.5 es una de las principales actualizaciones de la plataforma usada en dispositivos con tecnología Android. Dicha versión incluye nuevas funcionalidades, tanto para los usuarios como para los desarrolladores, así como cambios importantes en la estructura API de Android (nivel 3) [14]. Tiene la capacidad para grabar y reproducir video a través de la cámara. Desde el dispositivo, directamente se pueden subir los videos a YouTube o

[1] "*API Level*": Permite al sistema determinar correctamente si una aplicación es compatible con el sistema, antes de instalar la aplicación. Cada aplicación Android posee un valor entero especificado en "*android:minSdkVersion*", al igual que posee un nivel de API el SO del s*martphone*. Antes de instalar una aplicación, el sistema comprueba el "*API Level*" de dicha aplicación y permite la instalación si el valor es menor o igual al nivel de la API del propio sistema [14].

las fotos a Picasa. Posibilidad de personalizar los *widgets* mostrados en la pantalla de inicio. Inclusión del teclado en la pantalla con funcionalidades como la auto-corrección o la predicción del texto. Estas novedades, entre otras, empezaron a estar disponibles con la versión 1.5 que está basada en el *kernel* Linux 2.6.27.

Con respecto a la versión 1.6, conocida como DONUT, y que fue presentada no mucho más tarde que su predecesora, en Septiembre del 2009, se puede observar cómo está basada a su vez en un *kernel* distinto, el Linux 2.6.29. Esta nueva versión conlleva también algunas nuevas características, como por ejemplo, mejoras en la calidad de imagen, en la velocidad de la cámara y en la funcionalidad del Android Market. Posibilidad de borrar múltiples fotos a la vez en la galería y nueva funcionalidad para la conversión entre texto y voz. A pesar de todas estas novedades se considera una actualización menor, es decir, el salto dado entre esta nueva versión y la anterior no es tan pronunciado como fue el avance entre las versiones 1.1 y 1.5. Aumenta un nivel la API (Level 4) y posee 3 revisiones, la última de Mayo del 2010.

Unos meses más tarde de que saliese la v1.6, en Noviembre del mismo año, salió al mercado otra versión de Android con novedades, y de nombre ECLAIR y numeración 2.0. Una actualización de la misma tuvo lugar en Diciembre, 2.0.1, llegando hasta la versión 2.1 que se pudo utilizar en Enero del 2010. Esta actualización es de nuevo de tipo menor y se caracteriza principalmente por los cambios introducido en la API y la corrección de *bugs* (utiliza en este caso el nivel 7 dentro de la estructura API) [14]. Entre otras características se puede destacar el soporte para nuevos tamaños y resoluciones de pantallas, o las mejoras introducidas en el teclado virtual.

El último elemento de este grupo, la versión 2.2 apodada FROYO, tiene su origen en Mayo del 2010 (API Level 8), e introduce cambios en la API, correcciones de errores y posee nuevas características para los usuario y también para los desarrolladores [14]. Algunas de sus principales novedades son el aumento del rendimiento en los teléfonos móviles, el aumento de la memoria, la posibilidad de instalar aplicaciones en la memoria externa (*SD Card*), ó el aumento de la resolución de la cámara.

2.3.1.3 Características

Al ser una plataforma neutral de aplicaciones para dispositivos móviles, Android no diferencia entre las aplicaciones básicas del dispositivo y las implementadas por los distintos usuarios a la hora de establecer prioridades entre las aplicaciones. Esto hace posible que se puedan crear aplicaciones que sean tan importantes para el sistema operativo del teléfono como las que vienen ya de serie.

Los *background services* (código que se ejecuta en un segundo plano) permiten la creación de aplicaciones que usan un modelo orientado a eventos, trabajando en silencio cuando no se está manipulando el dispositivo o mientras se usan otras aplicaciones hasta que suena o vibra para llamar la atención. Android ofrece aplicaciones de mensajería P2P (*peer-to-peer*) entre dispositivos para mejorar la comunicación.

En la siguiente lista se destacan algunas de las características más notables de la plataforma Android [14]:

(1) ***Framework* de aplicaciones** permitiendo la reutilización y reemplazo de los componentes. Se provee al usuario de numerosos servicios para ayudarle a desarrollar aplicaciones.

(2) **Máquina virtual Dalvik** optimizada para dispositivos móviles. Android posee su propia máquina virtual para la gestión de las aplicaciones y su memoria.

(3) **Navegador integrado** basado en el motor de código abierto *WebKit*, facilitando así el acceso a Internet.

(4) **Gráficos optimizados** mediante una biblioteca de gráficos 2D, los gráficos 3D están basados en la especificación OpenGL ES 1.0 (aceleración de *hardware* opcional). Las pantallas ayudan a esta mejora en los gráficos al ser cada vez más grandes, brillantes y de alta resolución.

(5) **SQLite** para almacenamiento de datos estructurados. Se proporciona una base de datos relacional para cada aplicación mediante SQLite. Cada aplicación puede beneficiarse del motor que la gestiona para almacenar los datos de forma segura y eficiente.

(6) **Soporte multimedia** para formatos comunes de audio, vídeo e imágenes planas (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF). Se dispone de una amplia colección de bibliotecas que hacen posible dicho manejo multimedia.

(7) **Telefonía GSM** (dependiendo del *hardware*).

(8) **Bluetooth, EDGE, 3G y Wi-Fi** (dependiendo del *hardware*). La conectividad a Internet o entre distintos dispositivos depende del terminal que será el que determine los distintos protocolos de comunicaciones que soporta.

(9) **Cámara, GPS, brújula y acelerómetro** (dependiendo del *hardware*). Se incluyen bibliotecas API que simplifican el desarrollo de aplicaciones involucrando el *hardware* del dispositivo. Este cómodo acceso al *hardware* posibilita el hecho de no ser necesaria la creación de una implementación específica en el *software* para cada diferente dispositivo *hardware*.

(10) **Amplio entorno de desarrollo** incluyendo un emulador del dispositivo, herramientas para la depuración, perfiles de memoria y de rendimiento y un *plugin* para el Eclipse IDE (*Integrated Development Environment*).

2.3.1.4 Arquitectura

La arquitectura del SO Android se desglosa, al igual que cualquier otro sistema operativo, en distintas capas según su nivel de funcionalidad, Figura 2. 4. Un núcleo de Linux y una colección de bibliotecas de C y C++ se exponen a través del marco de trabajo de aplicaciones, que gestiona el entorno de ejecución y las aplicaciones [12].

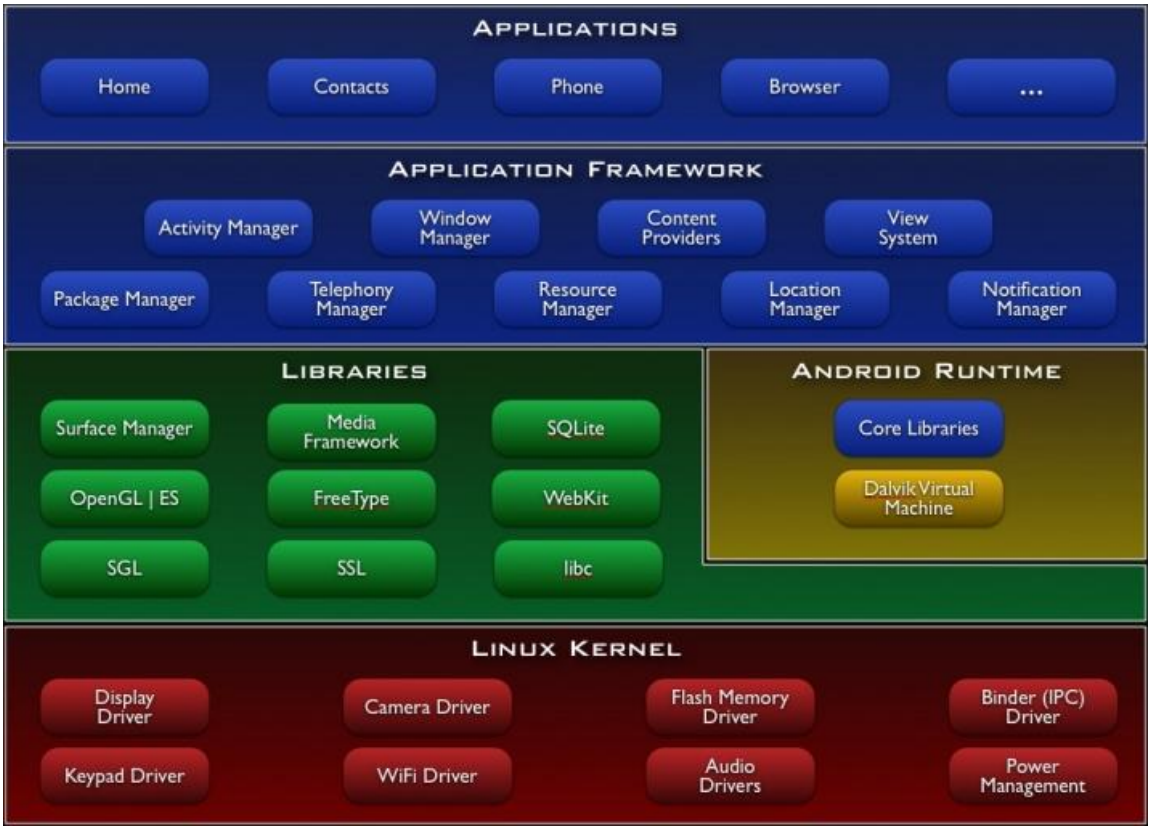


Figura 2. 4: Arquitectura de Android [14]

Los componentes de la arquitectura de Android mostrados en la Figura 2. 4, son explicados con un poco más de detenimiento a continuación:

Aplicaciones: Java es el lenguaje de programación en el que están escritas todas las aplicaciones. Entre las aplicaciones básicas se encuentra el navegador *Web*, cliente de *email*, mapas, contactos, programa de SMS o calendario. Las aplicaciones serán lo que los usuarios finales encuentren valioso sobre la plataforma Android, ya sean las que vienen preinstaladas o las descargadas del *market* de Android [15].

***Framework* de aplicaciones**: proporciona las clases necesarias para crear aplicaciones Android. Aporta también abstracción del acceso a hardware y gestiona la interfaz de usuario y los recursos de las aplicaciones. La arquitectura de Android impulsa el concepto de reutilización de recursos, permitiendo la publicación y compartición de actividades, servicios o datos entre aplicaciones a través de un acceso controlado (según las restricciones de seguridad que se impongan). Los desarrolladores tienen pleno acceso a las APIs usadas para implementar todo el sistema Android.

La base de todas las aplicaciones es un conjunto de servicios y sistemas que componen el citado marco de trabajo, entre ellos [14]:

- *Views*: conjunto extenso y variado de vistas que pueden ser usados en la realización de una aplicación, como son por ejemplo los botones, listas o cajas de texto.
- *Content Providers*: proveedores de contenido que permiten a las aplicaciones acceder a los datos de otras aplicaciones o compartir los suyos.
- *Resource Manager*: administrador de recursos que permite acceso a recursos relacionados con la estructura de datos, como las palabras claves externas, imágenes o archivos de *layout*.
- *Notification Manager*: administrador de notificaciones que permite a todas las aplicaciones mostrar alertas en la barra de estado.
- *Activity Manager*: administrador de actividades que gestiona el ciclo de vida de las aplicaciones y la interacción entre ellas.

***Runtime* de Android**: lo que hace diferente al teléfono Android de otros implementados en Linux, es el entorno de ejecución de Android. En éste se encuentra el conjunto base de bibliotecas y la máquina virtual Dalvik. El *Runtime* de Android es el motor que impulsa las aplicaciones [12].

- *Core Libraries*: proporcionan la mayoría de funcionalidades disponibles en las bibliotecas del lenguaje de programación Java.
- *Dalvik Virtual Machine*: máquina virtual basada en registros, que ha sido optimizada de forma que se pueda garantizar el que en un dispositivo se ejecuten múltiples instancias de forma eficiente.

Libraries: se incluyen un conjunto de bibliotecas escritas en C/C++ utilizadas por diversos componentes del sistema Android. Sus funcionalidades están a disposición de los desarrolladores mediante el uso del *framework* de aplicaciones de Android. Entre ellas se encuentran las siguientes bibliotecas [14]:

- LibWebCore: moderno motor de navegación *Web*.
- SGL: motor de gráficos 2D.
- FreeType: soporte para el manejo de distintos tipos de fuentes.
- SQLite: potente y ligero motor de base de datos relacionales disponible para todas las aplicaciones.

***Kernel* de Linux**: Android utiliza la versión 2.6 de Linux para sus principales funciones, como seguridad, administración de la memoria, administración de los procesos, pila de red y *drivers*. El núcleo actúa además como un nivel de abstracción entre el *hardware* y el resto de la pila de *software*.

2.3.1.5 Máquina virtual DALVIK

Dalvik es una máquina virtual especialmente diseñada para Android, desarrollada por Dan Bornstein y su equipo de Google. Desde sus inicios, Dalvik fue pensada para los dispositivos móviles, haciendo frente a las dos principales limitaciones que éstos conllevan, la duración de la batería y la potencia de procesamiento. El hecho de estar específicamente dirigida al entorno móvil la hace diferente de la máquina virtual de Java, que fue diseñada para ser una solución universal en todos los entornos [15].

Cada aplicación Android se ejecuta en su propio proceso, con su propia instancia de la máquina virtual Dalvik, que ha sido diseñada para poder ejecutar múltiples máquinas virtuales de manera eficiente en un mismo dispositivo. Está optimizada para hacer un uso mínimo de memoria gracias a utilizar los ficheros ejecutables con formato `.dex`. La máquina virtual está basada en registros y ejecuta las clases compiladas de Java, que mediante la herramienta incluida en el sistema "dx"(DX, *Dalvik Executable*) han sido transformadas al formato `.dex` (explicado en el siguiente párrafo e ilustrado en la Figura 2. 5). La máquina virtual Dalvik descansa sobre el núcleo de Linux que se encargará de las funcionalidades subyacentes, como puedan ser el tema de hilos o el manejo de la memoria a bajo nivel [14].

Las aplicaciones Android tienen su código fuente escrito en Java pero Dalvik, a diferencia de Java VM (Virtual Machine), no trabaja directamente con el *bytecode* de Java, sino que lo transforma en un código más eficiente que el original pensado para procesadores pequeños. En la Figura 2. 5 se observa como ambas máquinas virtuales parten por igual del código fuente, que una vez compilado da lugar al *bytecode* de Java. Éste será ejecutado en el caso de la Java VM, mientras que en el caso de la Dalvik VM, tendrá que ser recompilado para conseguir el *bytecode* de Dalvik [15].

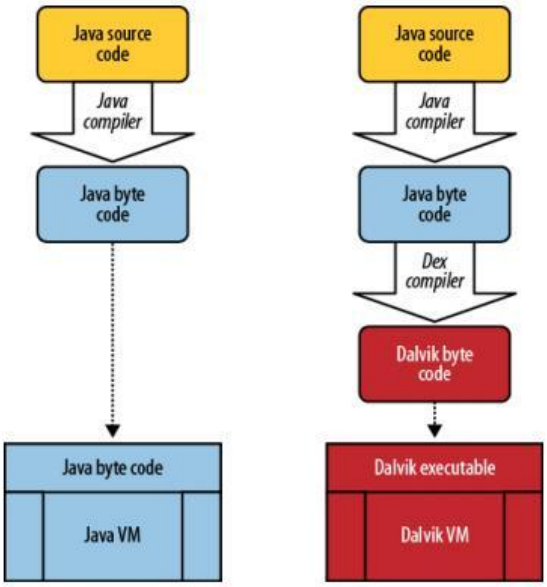


Figura 2. 5: Máquinas Virtuales, Java vs Dalvik [15]

Todo el *hardware* de Android y el acceso a servicios del sistema, se gestiona utilizando Dalvik como un nivel intermedio. Mediante el uso de una máquina virtual que se encargue de organizar la ejecución de aplicaciones, los desarrolladores tendrán un nivel de abstracción que garantiza que nunca tengan que preocuparse sobre una implementación *hardware* en particular [12].

2.3.2 Principales bloques de construcción de una aplicación Android

Los bloques de construcción básicos son los componentes que se usan en la implementación de las aplicaciones Android por parte de los desarrolladores. Estos componentes pueden depender unos de otros, pero cada uno de ellos existe como una entidad propia y desempeña un papel específico. Cada uno es una pieza única que ayuda, junto al resto de bloques, a definir el comportamiento global de la aplicación.

Existen cuatro tipos diferentes de componentes que una aplicación puede contener: *Activities*, *Services*, *Content providers* y *Broadcast receivers*.

2.3.2.1 Activity

Una actividad, por lo general, es una pantalla que el usuario ve en el dispositivo y con la que puede interactuar. Al ser la parte más visible de las aplicaciones Android, las actividades son el componente más utilizado. Cada una interactúa con los usuarios facilitándoles una ventana con la que puedan diseñar su propia interfaz de usuario.

Una aplicación normalmente está compuesta de múltiples actividades que están más o menos ligadas entre sí, y el usuario salta de una a otra. Una de las actividades de la aplicación es la denominada "*main activity*", es decir, la actividad principal, la que se ejecutará primero cuando se lance la aplicación. El movimiento de una pantalla a otra se logra al empezar una nueva actividad. Cada *activity* puede hacer que comience otra nueva a fin de realizar una acción diferente. El usuario podrá navegar hacia atrás o hacia delante fácilmente, puesto que cuando empieza una actividad, la anterior se detiene, pero el sistema la mantiene almacenada en una pila de actividades. Cuando comienza una actividad también se añade ésta a la pila. El hecho de usar dicha estructura de datos, hace posible que cuando el usuario quiere regresar a la pantalla previa, el sistema extrae y destruye la actividad actual y reanuda la anterior [14]. Relacionado con este tema está el apartado 2.3.3.3, en el que se expone el ciclo de vida de las actividades.

Para implementar una nueva actividad se creará una subclase de `Activity`, definiendo su propia interfaz de usuario (mediante una jerarquía de vistas, es decir, objetos derivados de `View`) y definiendo la nueva funcionalidad de la actividad [12]. En el apartado 2.3.3.4 se explica más detalladamente el tema de la interfaz de usuario.

Un ejemplo es una aplicación de correo electrónico que tenga una *activity* que muestre en la pantalla la lista con los nuevos *emails*, otra *activity* con la pantalla para poder escribir un nuevo correo y otra más para leer los mensajes.

2.3.2.2 Service

Un servicio es un componente que se ejecuta en un segundo plano (*background*) para realizar, o ejecuciones durante un largo periodo de tiempo, o trabajos para los procesos de control remoto. Es código no visible, no tiene interfaz gráfica [14]. Se usan para efectuar procesos periódicos que deben continuar aun cuando no estén activas o visibles las actividades de la aplicación.

Otro componente de la aplicación puede inicializar un *service* y éste continuará ejecutándose en *background* incluso cuando el usuario cambie a otra aplicación. En el apartado 2.3.3.3 se expone con más detenimiento el ciclo de vida de un *service*.

Los servicios son útiles para acciones que se quieren llevar a cabo durante un tiempo, sin importar que sea lo que aparece en la pantalla del dispositivo. Un ejemplo es una aplicación para reproducir música que sigue funcionando mientras el usuario se mueve de una aplicación a otra [15].

2.3.2.3 Content Provider

El proveedor de contenidos es el componente de Android encargado de hacer posible la compartición de datos entre aplicaciones. Por defecto, cada aplicación Android se ejecuta de manera independiente, almacenando sus propios datos en ficheros, bases de datos o cualquier otra estructura de almacenamiento accesible por ella, haciendo que todos sus datos permanezcan aislados del resto de aplicaciones del sistema. La necesidad de intercambiar información entre aplicaciones se solventa a través de los *content providers* que, dependiendo de los permisos establecidos, permitirán que una aplicación, consulte, añada, borre o actualice datos de cualquier otra.

Como ejemplo de proveedor de contenidos se encuentra el facilitado por el sistema Android para gestionar la información de los contactos del usuario. Cualquier aplicación que posea los permisos adecuados puede, por tanto, acceder a la información sobre alguna persona en particular [14].

2.3.2.4 Broadcast Receiver

Es el componente destinado a detectar y reaccionar ante los mensajes *broadcast* que se producen en el sistema, ya sean originados por el propio sistema o por cualquier otra aplicación. El receptor es un código inactivo pero que, una vez que se lanza un evento al cual está suscrito, se activa.

Los *broadcast receivers* no tienen ninguna interfaz gráfica asociada, pero para informar al usuario de que ha ocurrido algún evento externo, pueden crear notificaciones o alertas que se muestran en la barra de estado (ver ejemplo de la Figura 2. 6).

Algunos ejemplos que escenifican este componente son los anuncios de que la pantalla se ha apagado, la batería está baja o informar a otras aplicaciones de que se ha descargado algún dato en el dispositivo y éste está disponible por su uso [14].

Figura 2. 6: Ejemplo de icono de notificación en la barra de estado [14]

2.3.3 Conceptos importantes

A continuación se van a exponer algunos de los fundamentos más relevantes de la plataforma Android.

2.3.3.1 Aplicaciones, Tareas, Procesos e Hilos

Existen diversos términos a la hora de entender el funcionamiento de una aplicación Android que pueden llevar a equívoco. Es por eso que se va a proceder a identificarlos y ver la relación que los unos guardan con los otros.

Android es diferente de la mayoría de los sistemas operativos, favoreciendo el que, a pesar de no ser lo mismo, se confundan los términos aplicación y proceso. En Android, una aplicación puede estar viva a pesar de que su proceso haya muerto. Es decir, el ciclo de vida de una actividad no está atado al de un proceso. Los procesos son solo contenedores disponibles para las actividades [16].

Aplicación: una aplicación Android consiste en la unión de uno o más componentes (típicamente actividades) que interactúan con el usuario y está envuelto en un archivo característico, un paquete (fichero con terminación `.apk`).

Cuando un bloque de una aplicación comienza y ésta no tiene ningún otro componente ejecutándose, el sistema Android inicia para dicha aplicación un nuevo proceso (Linux *process*) con un único hilo de ejecución. Por defecto, todos los componentes de una misma aplicación se ejecutan en el mismo proceso e hilo (hilo principal, "*main thread*"). Sin embargo, es posible planificar que algunos elementos se ejecuten en procesos separados y que se puedan crear hilos adicionales.

Tarea: una tarea es la secuencia de actividades que el usuario sigue para lograr un objetivo, independientemente de las aplicaciones a las que pertenezcan dichas actividades. Hasta que no se especifica explícitamente una nueva tarea, todas las actividades que inicia el usuario se consideran parte de la tarea actual.

Cabe destacar que las actividades de una tarea pueden ser de cualquier aplicación, es decir, pueden pertenecer a distintas aplicaciones o todas a la misma. La actividad que da comienzo a una tarea se denomina actividad raíz, "*root activity*".

Actividad: las actividades (apartado 2.3.2.1) son el bloque de construcción básico de las aplicaciones de Android. Una aplicación se puede formar con actividades nuevas y/o reutilizar actividades de otras aplicaciones. Todas ellas trabajarán en conjunto para formar una interfaz de usuario coherente.

Cualquier aplicación que presente algo en la pantalla debe tener al menos una actividad responsable de dicha visualización.

Proceso: normalmente existe un único proceso para cada aplicación en el que se ejecutan todos sus elementos. Si es necesario controlar el proceso con respecto a un componente, se puede hacer en el fichero AndroidManifest.xml (ver apartado 2.3.3.5) indicándole al componente (*Activity, Service, Content Provider, Broadcast Receiver*) el atributo `android:process`.

Hilo: cada aplicación posee un hilo de ejecución que es creado por el sistema al lanzarse la aplicación. Aunque lo común es un hilo en cada aplicación, se permite a los

desarrolladores crear más. Si la aplicación realiza alguna intensa labor, (largas operaciones como puedan ser el acceso a la red o consultas a las bases de datos), el modelo de un único hilo puede provocar un mal rendimiento si no se implementa la aplicación apropiadamente [14].

2.3.3.2 Ciclo de vida de una aplicación Android

El ciclo de vida de una aplicación se constituye según los pasos que deban seguir sus procesos desde el comienzo hasta la finalización de la aplicación. Todas las aplicaciones, independientemente del lenguaje de programación en el que estén escritas, tienen un ciclo de vida especifico, y las aplicaciones Android no son ninguna excepción.

La peculiaridad del ciclo de vida de una aplicación Android reside en el hecho de que el tiempo de vida de un proceso no es controlado directamente por la aplicación. En lugar de esto, el sistema es el que controla la mayor parte de las veces, el tiempo de vida de las aplicaciones. Todas las aplicaciones se ejecutan en sus propios procesos, y éstos son controlados por el sistema de Android que, dependiendo de las necesidades, puede terminar algún proceso para liberar recursos [17].

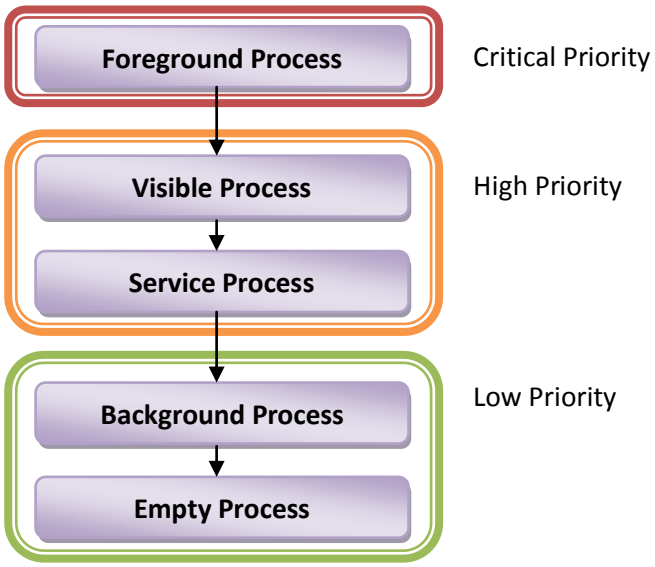


Figura 2. 7: Niveles de “importancia” de los procesos

El sistema siempre trata de mantener los procesos de las aplicaciones el mayor tiempo posible, pero de vez en cuando, necesita eliminar procesos antiguos para dejar memoria suficiente a los más importantes o nuevos procesos. Para determinar qué procesos mantener y de cuales prescindir, el sistema coloca cada proceso dentro de una jerarquía según su importancia (Figura 2. 7). Dicha estructura se basa en los componentes que se estén ejecutando en los procesos y el estado de esos componentes. Los procesos de menor importancia serán los primeros eliminados, después serán los siguientes de importancia más baja y así sucesivamente hasta que deje de ser necesario matar procesos para recuperar los recursos del sistema.

Hay cinco niveles diferentes en la jerarquía de importancia. En la Tabla 2. 1 se presentan todos los posibles procesos según dicha estructura [14].

	Niveles de "importancia"
+	*Foreground Process*
|	*Visible Process*
|	*Service Process*
▼	*Background Process*
-	*Empty Process*

Tabla 2. 1: Tipos de procesos según su nivel de prioridad

- ***Foreground Process***: proceso de primer plano, necesario para la acción que desarrolla el usuario en la actualidad. Coexistirán pocos de este tipo al mismo tiempo en el sistema. Serán eliminados como último recurso si la memoria es tan baja que ni ellos mismos pueden continuar ejecutándose.

- ***Visible Process*****:** proceso que no tiene ningún componente en primer plano, pero sí puede afectar a lo que el usuario está viendo por pantalla. Son procesos de una gran importancia, por lo que no serán eliminados a menos que sea necesario para que los *foreground process* puedan seguir ejecutándose.

- ***Service Process*****:** proceso que ejecuta un *service*, por lo que no afecta directamente a lo que se muestra por pantalla al usuario, pero sí ejecuta otras acciones requeridas por éste (reproducción de música o descarga de datos). El sistema siempre los mantendrá en ejecución a no ser que no haya memoria suficiente para que éstos funcionen junto con los dos procesos de niveles superiores.

- ***Background Process*****:** proceso que ejecuta una *activity* no visible para el usuario en la actualidad. Al no tener repercusión directa en las acciones que está desarrollando el usuario, el sistema puede eliminar estos procesos en cualquier momento que se necesite memoria para alguno de los tres tipos de procesos superiores. El criterio lo establece una lista según la cual, las últimos procesos con *activities* que hayan sido mostradas por pantalla serán los últimos en ser eliminados, lista LRU (*Least Recently Used*).

- ***Empty Process*****:** proceso que no aloja ningún bloque de construcción activo. Su utilidad es mejorar el tiempo de arranque la próxima vez que el componente tenga que ejecutarse en él. El sistema los elimina a fin de equilibrar los recursos existentes de memoria.

El sistema de Android clasifica los procesos de forma que éstos se sitúen en el mayor nivel posible. Esta catalogación se basa siempre en la importancia que tengan los componentes activos presentes en el proceso.

2.3.3.3 Ciclos de vida de los principales componentes de una aplicación

Entender correctamente los ciclos de vida de los bloques de una aplicación es vital para garantizar que la aplicación proporcione una perfecta experiencia al usuario, a la vez que se gestionan adecuadamente los recursos del sistema [12]. Cada uno de los componentes básicos de las aplicaciones Android tiene su propio ciclo de vida. En este apartado se van a exponer los ciclos de vida de dos de los principales componentes: *Activies* y *Services*.

Ciclo de vida de una *Activity*

La gestión del ciclo de vida de las actividades mediante la implementación de los correspondientes métodos (Tabla 2. 2) es fundamental a la hora de lograr una aplicación robusta y flexible. El ciclo de vida de una *activity* depende directamente de su asociación con las demás y de la pila de actividades, "*back stack*" [14].

El estado de cada *activity* se determina según sea su posición en la pila de actividades, siendo ésta la estructura que recoge todas las actividades que están actualmente en ejecución según el criterio "último en llegar, primero en salir" [12].

Los estados viables en los que puede estar una actividad son los tres siguientes:

Activa (*Resumed*): la actividad se encuentra en un primer plano de la pantalla, está visible e interactuando con el usuario. En este estado la *activity* está en la parte más alta (primera posición) de la pila de actividades.

En pausa (*Paused*): la actividad está visible en la pantalla pero no es la que interactúa con el usuario. Es decir, otra actividad activa, parcialmente transparente o que no ocupa toda la pantalla, está por encima de ésta.

Parada (*Stopped*): la actividad se encuentra oculta por otras actividades, no está visible en el dispositivo.

El ciclo de vida de una actividad, es decir, los posibles caminos que una actividad puede recorrer entre los diferentes estados, se puede observar en la Figura 2. 8 [14].

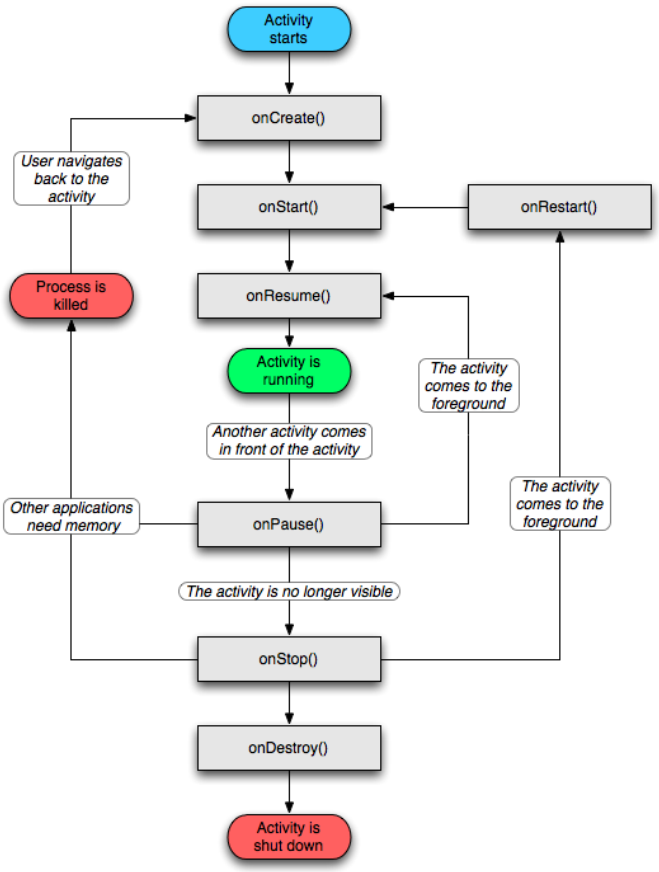


Figura 2. 8: Ciclo de vida de una *Activity* [14]

Los métodos necesarios para manejar la transición de las actividades entre estados, y que en la Figura 2. 8 se representan en forma de rectángulos, se detallan en la Tabla 2. 2. Estos métodos: *onCreate(), onRestart(), onStart(), onResume(), onPause(), onStop() y onDestroy(),* son necesarios para definir el ciclo de vida completo de una actividad y controlar los bucles anidados que se forman (Figura 2. 9). Son tres los bucles que controlan y manejan los diferentes tiempos de vida:

Tiempo de vida total (*entire lifetime*): todo lo que le sucede a una actividad entre la llamada al método `onCreate()` y la llamada a `onDestroy()`.

Tiempo de vida visible (*visible lifetime*): todo lo que le ocurre a una actividad entre la llamada al método `onStart()` y la llamada a `onStop()`. Durante este periodo de tiempo, el usuario puede ver la actividad en la pantalla e interactuar con ella o no, según el estado en el que se encuentre la actividad (activa o en pausa). Dentro del tiempo de vida total de la aplicación pueden existir múltiples *visible lifetime*, ya que las actividades pueden alternar entre estar oculto o visibles.

Tiempo de vida activo (*foreground lifetime*): tiempo de una actividad que transcurre entre la llamada al método `onResume()` y la llamada a `onPause()`. En este intervalo de tiempo, la actividad se sitúa en el primer plano de la pantalla mostrándose por delante de todas las demás actividades que puedan estar también dentro de la pantalla y es la que interactúa con el usuario.

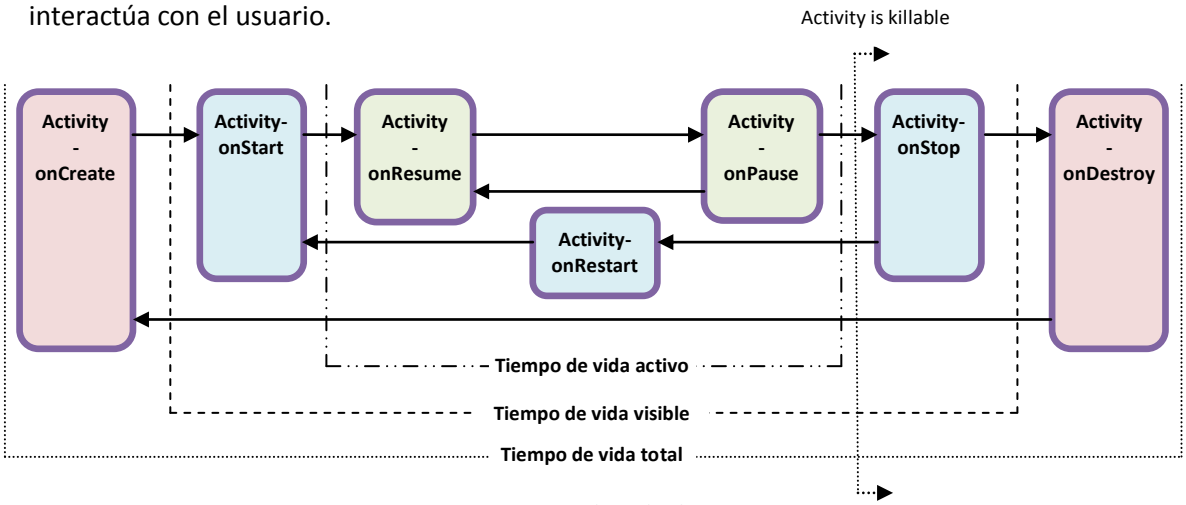


Figura 2. 9: Tiempos de vida de una *Activity*

Método	Descripción	Eliminable?	Próximo método
onCreate()	Se llama cuando se crea la actividad por primera vez.	No	`onStart()`
onRestart()	Llamado después de que se haya parado la actividad y justo antes de volver a comenzarla.	No	`onStart()`
onStart()	Se invoca cuando se quiere que la actividad sea visible para el usuario.	No	`onResume()` o `onStop()`
onResume()	Actúa justo antes de que la actividad comience a interactuar con el usuario.	No	`onPause()`
onPause()	Método llamado cuando el sistema va a ejecutar otra actividad en su estado activo.	Si	`onResume()` o `onStop()`
onStop()	Se llama a éste método cuando la actividad deja de estar visible para el usuario.	Si	`onRestart()` o `onDestroy()`
onDestroy()	Método invocado antes de que la actividad sea destruida. Será la última llamada que reciba la actividad.	Si	-

Tabla 2. 2: Métodos del ciclo de vida de una Actividad

Ciclo de vida de un *Service*

Los servicios tienen un ciclo de vida mucho más sencillo que las actividades, un servicio comienza o se para. En este caso el control del ciclo de vida del *service* depende más de los desarrolladores de aplicaciones que del sistema Android. Por consiguiente, al desarrollar una aplicación Android se debe tener en cuenta que al ejecutar los servicios no se consuman recursos compartidos de manera innecesaria, como puedan ser la batería o la CPU [15].

Aunque este ciclo de vida sea simple, es de suma importancia prestar atención a cómo se crea y se destruye un servicio. La razón para ello es que un servicio puede estar ejecutándose en un segundo plano sin que el usuario sea consciente de ello.

El ciclo de vida de un servicio, desde que se crea hasta que se destruye (Figura 2. 10), puede tomar dos caminos distintos [14] según su inicialización:

Iniciado con startService: se crea el servicio a partir de la llamada al método `startService()` desde otro componente. El servicio se ejecutará indefinidamente, hasta que el servicio mismo llame a la función `stopSelf()`, o hasta que otro componente llame a `stopService()`. Cuando esto pase, el *service* se parará y el sistema lo eliminará.

Iniciado con bindService: el cliente llama a la función `bindService()` para crear el servicio y se comunica con éste a través de la interfaz `IBinder`. Para cerrar la comunicación con el servicio, el cliente utiliza el método `unbindService()`. Muchos clientes pueden conectarse a un mismo servicio, por lo que el sistema eliminará el servicio cuando todos los clientes hayan cortado su comunicación con él.

Estas dos opciones no son excluyentes, es decir, un cliente se puede conectar a un servicio que ya haya sido inicializado con `startService()`. Sin embargo, hasta que el cliente no se desligue del servicio, los métodos `stopService()` o `stopSelf()` no podrán pararlo.

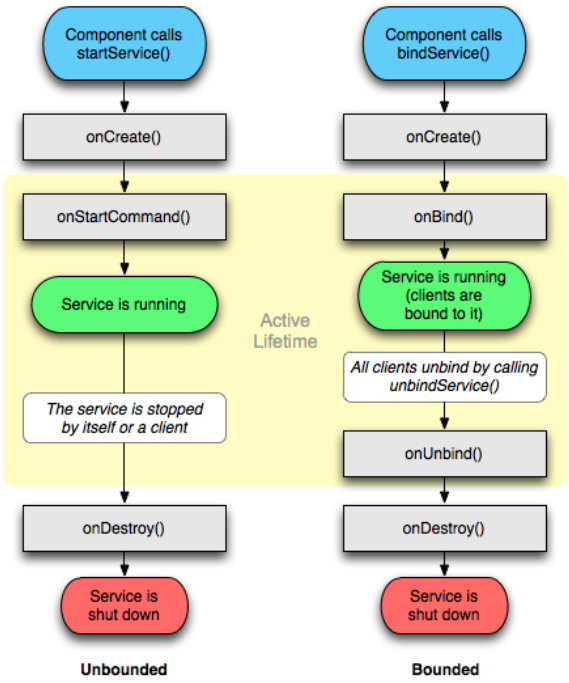


Figura 2. 10: Ciclo de vida de un *Service* [14]

2.3.3.4 Interfaz de usuario

En una aplicación Android, la interfaz de usuario, UI (*User Interface*), se implementa a través del uso de objetos *View* y *ViewGroup*. Existen muchos tipos diferentes de ambos, aunque cada uno de ellos hereda siempre de la clase `View`. Esta clase se usa como base para las subclases que ofrecen objetos totalmente implementados para el control visual, llamadas *widgets* [14].

View (vista): estructura de datos cuyas propiedades permiten establecer los parámetros del *layout* y el contenido de una zona rectangular especifica de la pantalla. Un objeto *View* es la unidad básica de expresión de la interfaz de usuario en la plataforma Android. Una vista controla sus propios elementos, como medidas o diseño, dentro del área rectangular de la pantalla donde se encuentra. Al ser un objeto de la interfaz de usuario, *View* es también un punto de interacción para los usuarios, a la vez que receptor de eventos interactivos.

ViewGroup (grupo de vistas): objeto especial de *view* que contiene dentro de él a otras vistas, a las que se les llama hijos. Cada grupo de vistas encierra un conjunto de reglas sobre como mostrar sus propios elementos hijos *views*.

Jerarquía de *View*

En la plataforma Android, se define la UI de una *activity* usando una jerarquía de vistas y grupo de vistas, en concreto se sigue un diagrama de árbol, Figura 2. 11. Este árbol estructurado puede ser tan simple o tan complejo como sea necesario. Se puede desarrollar mediante el conjunto predefinido de Android de *widgets* y *layout*, o a través de la creación de nuevas *views* que se construya el desarrollador.

Dentro de esta estructura, los objetos *views* son las hojas del árbol y los objetos *viewgroups* son las ramas.

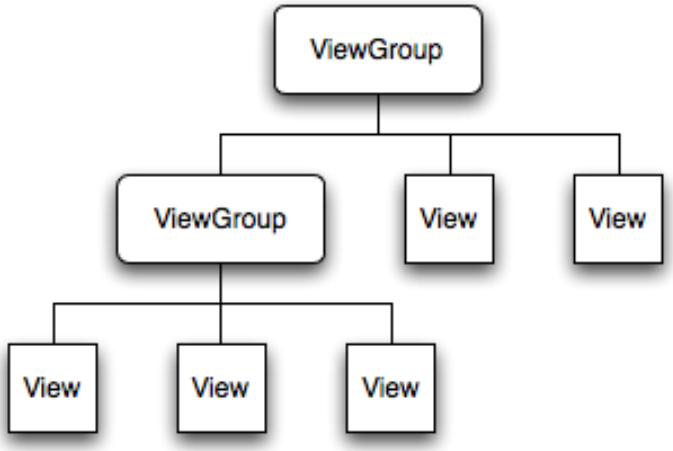


Figura 2. 11: Árbol estructurado de *Views* [14]

Para que esta estructura, árbol jerarquizado de vistas, Figura 2. 11, se muestre por pantalla del dispositivo, se procede de la siguiente manera:

La *activity* invoca a la función `setContentView()` pasando como referencia el nodo raíz del árbol. Una vez que ésta referencia se recibe por el sistema Android, se procede entonces a medir y dibujar el árbol. El nodo raíz de la estructura indica a sus hijos que se dibujen, siendo cada *viewgroup* responsable de sus elementos hijos [14].

Layout

Un fichero *layout* XML (*Extensible Markup Language*) es la forma más común de definir el diseño de la organización de *views*. Este archivo tiene una estructura legible, donde cada elemento es un objeto *view*, *viewgroup* o algún descendiente de ellos.

Hay numerosas formas de realizar los diseños a través de múltiples tipos de *views* y *viewsgroups*. Algunos de los *viewgrops* predefinidos ya por Android, los *layouts*, son *LinearLayout, RelativeLayout, TableLayout y GridLayout*, cada uno de ellos ofrece un conjunto único de parámetros de diseño, que se utilizan para definir la posición y características de sus elementos.

Widgets

Un *widget* es un objeto *view* que actúa como interfaz para interactuar con el usuario. La plataforma Android proporciona un amplio conjunto de *widgets* ya implementados, como es el caso de botones, marcadores o campos de texto, con los que construir rápida y fácilmente una interfaz de usuario. Algunos de estos *widgets* que facilita el sistema son algo más complejos, caso del reloj o del selector de fechas.

Opciones de implementación

A la hora de llevar a cabo la implementación de la UI de Android hay dos opciones diferentes, la primera es la forma esquemática a través de los ficheros *layouts* (información más detallada en el apartado 2.3.3.6), y la segunda mediante la programación de código, ambas son distintas pero normalmente son usadas conjuntamente para realizar un buen trabajo.

La primera opción define (en los archivos XML) como se verá la interfaz de usuario, los elementos que aparecerán en la pantalla, y la segunda implica desarrollar la UI a través de escribir código Java, es decir, declarando, instanciando y seleccionando las propiedades y funcionalidades de cada elemento.

La mejor solución por tanto, a la hora de realizar la UI, es usar las dos formas. Se suele declarar en el *layout* de XML todo lo que sea estático de la interfaz de usuario, y en el código Java todo lo dinámico, lo que interactúe con el usuario. En otras palabras, se usa XML para declarar dónde está y cómo es el 'botón', y en Java se especifica qué hace [15].

2.3.3.5 Fichero AndroidManifest.xml

El `AndroidManifest.xml` es un fichero obligatorio para cualquier aplicación Android, todos los proyectos deben tenerlo en su carpeta principal, y en él definir la estructura de la aplicación y de sus componentes.

Este archivo contiene información muy relevante de la aplicación, necesaria para que el sistema pueda ejecutar el código de la aplicación. Es decir, antes de que el sistema Android pueda iniciar un componente de la aplicación, éste debe existir en el archivo `AndroidManifest.xml`.

En la Tabla 2. 3 se enumeran algunas de las principales funciones del archivo del proyecto Android, `AndroidManifest.xml`.

Función	Código
Nombrar el paquete Java de la aplicación	`package="com.android.project"`
Descripción de los componentes de la aplicación	`<activity …/> <service .../> <receiver …/> <provider …/>`
Determinar los procesos que acogerán a los componentes	`android:process`
Declarar los permisos que tendrá la aplicación al interactuar con otras aplicaciones o con las partes protegidas de la API	`<uses-permission android:name="…"/>`
Declarar los permisos que los demás necesitarán para interactuar con los componentes de la aplicación	`android:permission`
Enumerar las clases `Instrumentation` que aportan información al probar la ejecución de la aplicación	`<instrumentation android:name="…"/>`
Declarar el nivel mínimo del API de Android que se requiere	`<uses-sdk android:minSdkVersion="8"/>`
Enumerar las bibliotecas	`<uses-library Android:name="…" />`

Tabla 2. 3: Ejemplos de funciones del fichero AndroidManifest.xml

De la Tabla 2. 3 cabe destacar la importancia de la descripción de los componentes (*activities, services, broadcast receivers* y *content providers*) de los que consta la aplicación. Se nombran las clases que implementarán cada uno de los componentes y se publicarán sus características. Todo esto permite que el sistema sepa qué bloques existen y bajo qué condiciones deben funcionar.

En el apartado 2.3.3.8 se puede entender mejor la finalidad de los elementos relacionados con los permisos y la seguridad de la aplicación Android.

2.3.3.6 Fichero main.xml

El *layout* es la arquitectura de la UI de una actividad. En él se define el diseño de la estructura y los distintos elementos que se mostrarán por pantalla al usuario.

El fichero *layout* XML se suele llamar por lo general `main.xml` y se encuentra situado en la carpeta `res/layout` del proyecto. Además de este fichero, puede coexistir algún otro archivo de *layout* en la carpeta, ya que una misma aplicación puede mostrar distintas distribuciones de pantalla.

El fichero esquemático XML de *layout* ofrece un amplio espectro de oportunidades de diseño a la hora de desarrollar una aplicación. El archivo proporciona una interfaz (sin necesidad de programar código en la propia actividad) para el diseño y estructura de la pantalla del dispositivo. En estos ficheros XML también se incluyen las herramientas necesarias para que los desarrolladores de aplicaciones Android, en tiempo de ejecución, puedan modificar parámetros de los elementos de diseño [18].

En `main.xml`, o en cualquier otro fichero *layout*, se debe tener en cuenta que éste debe presentar un elemento raíz (objeto de tipo *view* o *viewgroup*), y una vez que se tiene definido el nodo principal *(Layout*) ya se pueden añadir los elementos adicionales (*layouts, widgets*) que serán los nodos hijos. Con todo esto se forma la jerarquía de vistas (Figura 2. 11) que define el diseño que presentará la aplicación.

2.3.3.7 Fichero R.java

El archivo `R.java` es un fichero generado automáticamente por el sistema Android y que se añade a la aplicación cuando se crea el proyecto. Este fichero contiene los punteros a los directorios de recursos como *drawable, layout o values*, y a los elementos que están dentro de dichos directorios. Es un fichero que no debe modificarse directamente por los desarrolladores sino que se va actualizando él mismo cuando se hace alguna modificación en los recursos [17].

El concepto de 'recurso' se aplica a una imagen, cadena de texto específica, archivo *layout* o cualquier otra información que no sea parte del programa en código Java que necesite la aplicación. Todos los recursos que se crean y utilizan, deben estar guardados en la carpeta del proyecto llamada `res`. El sistema Android procesará y compilará los recursos según la subcarpeta en la que se encuentren (*drawable*, *layout*, etc) y el formato de los ficheros [16].

El archivo `R.java` es el nexo de unión entre el mundo de Java y el mundo que forman los recursos del proyecto Android.

2.3.3.8 Seguridad

La mayor parte de las medidas de seguridad que se pueden encontrar en la plataforma Android provienen del *kernel* de Linux, como ya se mencionó en el apartado 2.3.1.4. Cada aplicación se ejecuta en su propio proceso de Linux. El propio *hardware* prohíbe que un proceso acceda a la memoria de otro proceso. Por otra parte, a cada aplicación se le asigna un identificador específico. Los archivos que genere no podrán ser leídos ni sobrescritos por ninguna otra aplicación.

Además de todo esto, el acceso a determinadas operación críticas es restringido. Por lo tanto, para poder usar la aplicación se debe especificar en el fichero `AndroidManifest.xml` los permisos que se requieren (`<uses-permission … />`). Cuando se instala la aplicación en el dispositivo, el gestor "*Package Manager"* concede o deniega los permisos solicitados en base a los certificados, e incluso si es necesario, preguntando al usuario por su conformidad a la hora de otorgar dichos permisos.

A su vez Android puede restringir el acceso por parte de terceros, a partes enteras del sistema. A través, de nuevo en el fichero `AndroidManifest.xml`, de etiquetas XML (`android:permission`) se puede dictaminar quien puede iniciar una *activity*, conectarse o iniciar un *service*, recibir *broadcast,* o acceder a los datos de un *content provider* [16].

2.3.3.9 Almacenamiento de datos

Android proporciona diversas opciones a la hora de almacenar los datos de una aplicación de forma persistente, la solución que se tome en cada caso será dependiente de las necesidades específicas que tenga cada aplicación, es decir, se tendrán en cuenta aspectos importantes, como por ejemplo el hecho de que los datos sean privados o por el contrario deban estar accesibles a otras aplicaciones, o por ejemplo el espacio que se necesita para guardar los datos.

Guardar y cargar datos es un requisito esencial para la mayoría de las aplicaciones. Como mínimo, las actividades deben almacenar el estado de la UI cada vez que sean

desplazadas y dejen de estar en primer plano. Esto es necesario para garantizar que, cuando ésta vuelva a aparecer en pantalla, se muestre la interfaz de usuario en el mismo estado en que se encontraba anteriormente.

Otro caso en el que también se necesita almacenar datos en una aplicación Android, es a la hora de guardar las preferencias de los usuarios para que éstos puedan personalizar las aplicaciones [12].

En la Tabla 2. 4 se resumen las distintas opciones que presenta Android para almacenar los datos de las aplicaciones según las necesidades.

Mecanismo de almacenamiento	Descripción
Shared Preferences	Mecanismo simple y ligero del tipo contraseña-valor para guardar datos primitivos de las aplicaciones, comúnmente las preferencias de los usuarios.
Almacenamiento interno	Almacenamiento de datos privados en ficheros dentro de la memoria del dispositivo.
Almacenamiento externo	Almacenamiento de datos públicos en archivos dentro de memorias externas compartidas.
Bases de datos SQLite	Almacena datos estructurados de forma robusta en una base de datos privada sobre la que se tiene un control total.
Conexión red	Almacenamiento de datos en la *web* con un servidor de red.
Content Providers	Componente que permite, según las restricciones que se establezcan, acceso (de lectura o escritura) a los datos de la aplicación. Ver apartado 2.3.2.3.

Tabla 2. 4: Mecanismos de almacenamiento de datos

2.3.3.10 Organización de un proyecto Android

Todas las aplicaciones Android a nivel de desarrollador, se entienden como proyectos que se organizan según un determinado sistema de ficheros y directorios (ver Figura 2. 12).

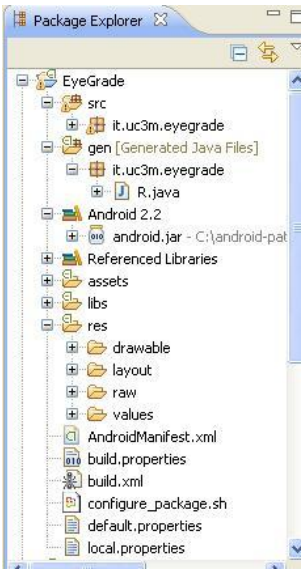


Figura 2. 12: Estructura de un proyecto Android

La estructura del proyecto (ejemplo presentado en la Figura 2. 12) tiene un directorio raíz del mismo nombre que el proyecto, éste es la sede o sitio de depósito de todos los archivos del proyecto, ya sean ficheros creados por el desarrollador o archivos generados automáticamente por el sistema. Los tres principales elementos que hay en el citado directorio son: el fichero `AndroidManifest.xml` (explicado en el apartado 2.3.3.5), el paquete de bibliotecas *Referenced Libraries*, y una serie de directorios (*src, gen, assets* y *res*) [17].

Los paquetes de bibliotecas contienen todos los archivos `.jar`, es decir, todas las bibliotecas a las que se hace referencia por parte del proyecto de la aplicación Android. Inicialmente todos los proyectos parten con una única biblioteca, propia de la versión Android que se utilice, que se llama `android.jar` y que contiene las funciones de la API de Android. A parte de esta biblioteca también se pueden añadir bibliotecas Java externas al sistema operativo.

También hay cuatro directorios dentro de la raíz del proyecto y cada uno de ellos con un objetivo diferente. Estas carpetas juegan un rol muy importante en el desarrollo de la aplicación. La Tabla 2. 5 muestra de forma esquemática los directorios más importantes de toda aplicación Android.

Directorio	Subcarpeta	Finalidad
src		Contiene todos los archivos de código fuente de Java. Se almacenan con un formato relativo al nombre del paquete de la aplicación, como sería: `src/your/package/namespace/ActivityName.java`
gen		Almacena el archivo autogenerado `R.java` (Ver apartado 2.3.3.7).
assets		Directorio que puede guardar cualquier tipo de fichero externo que sea necesario para la aplicación.
res		Contiene los recursos de la aplicación, todo lo que no sea código Java pero que debe aparecer en la aplicación.
res/	**drawable**	Carpeta donde se encuentran las imágenes que usa el proyecto. Como mínimo el icono de la aplicación `icono.jpg`.
res/	**layout**	Carpeta donde se encuentran los ficheros *layout* XML como el `main.xml` (Ver apartado 2.3.3.6).
res/	**raw**	Carpeta usada para almacenar el resto de ficheros externos. Similar a la carpeta *assets* pero con la diferencia de que se accede a los datos a través de los identificadores de recursos de la clase `R`.
res/	**value**	Carpeta que contiene archivos XML como `string.xml`, para guardar las constantes de tipo *String* o *Array* que se muestren en la aplicación.

Tabla 2. 5: Directorios de un proyecto Android

A partir de los proyectos Android se construyen los ficheros `.apk` que son los que se instalan en los dispositivos, en ellos se almacena toda la información de la aplicación, tales como el código fuente o los recursos de la aplicación.

2.3.3.11 Herramientas de desarrollo

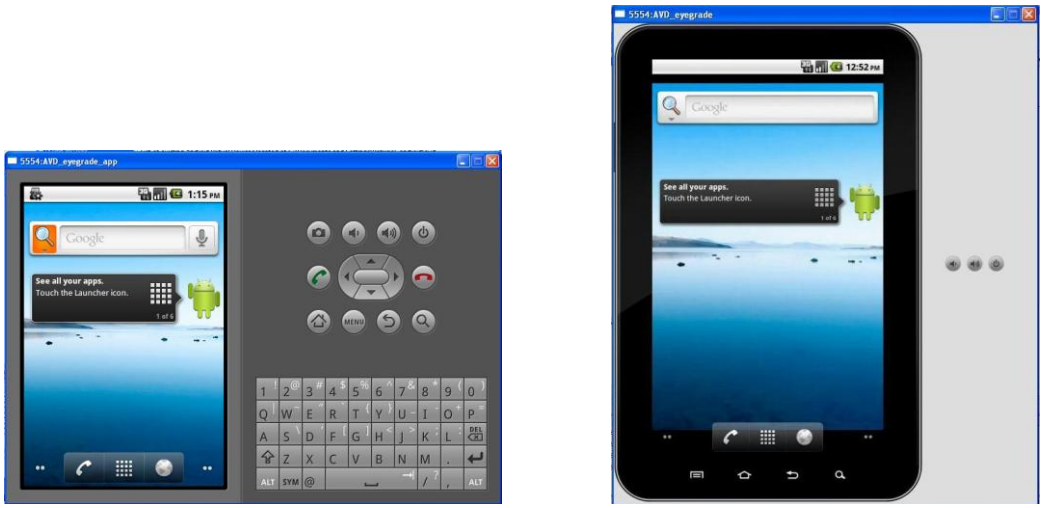
Las herramientas necesarias para el desarrollo de aplicaciones sobre la plataforma Android, se basan en el entorno de desarrollo Eclipse[2] (ver apartado 5.1) y en el *plugin* de Google relacionado con dicho entorno de desarrollo, de nombre ADT (*Android Development Tools*) o herramienta de desarrollo para Android. ADT es un *plugin* para el IDE de Eclipse que está diseñado para ofrecer un entorno potente e integrado, en el que se puedan realizar las aplicaciones Android.

ADT amplia el potencial de Eclipse facilitando:

- La rápida configuración de un nuevo proyecto Android.
- La creación de la interfaz de usuario de una aplicación.
- El añadir componentes basados en la Android *Framework* API.
- La exportación de los ficheros `.apk` (firmados ó no) para la distribución de la aplicación.
- La depuración de las aplicaciones mediante la utilización de las herramientas del SDK (*Software Development Kit*) de Android.

El SDK de Android (Kit de desarrollo *software* para Android) incluye una gran variedad de herramientas útiles, para el desarrollo de aplicaciones móviles sobre la plataforma Android. Las herramientas se pueden clasificar en los siguientes dos grupos [14]:

SDK *tools* (herramientas SDK): independientes de la versión Android utilizada. Se instalan con el paquete inicial SDK, se actualizan periódicamente y son obligatorias para poder desarrollar aplicaciones Android. Una de las herramientas más importantes de este conjunto es el emulador. En la Figura 2. 13 se pueden observar dos ejemplos de emuladores, cada uno de ellos con un tipo de pantalla especifico. El primero posee una pantalla HVGA (*Half-size Video Graphics Array*) con resolución 320x480 pixeles y el segundo una pantalla correspondiente al modelo de la tableta *Galaxy* de 600x1024. En la Tabla 2. 6 se resumen las herramientas utilizadas con mayor frecuencia.



(a) Pantalla HVGA (b) pantalla GALAXY TAB

Figura 2. 13: Emuladores de dispositivos móviles Android

[2] Existen otros entornos de desarrollo diferentes, pero Eclipse es el más generalizado y el usado a lo largo de todo este proyecto fin de carrera.

Platform tools (herramientas según la versión Android): herramientas adaptadas para dar soporte a las características de la última versión de Android. Se actualizan cada vez que se instale una nueva versión del SDK. Cada actualización de este tipo de herramientas es compatible con las versiones más antiguas. Estas herramientas se presentan en la Tabla 2. 7.

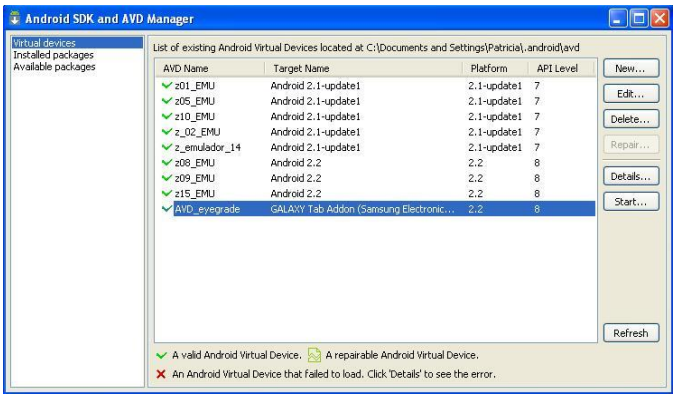


Figura 2. 14: AVD Manager

Herramienta	Descripción
android	Importante herramienta que permite la gestión de dispositivos virtuales AVDs (*Android Virtual Devices*), proyectos y componentes instalados del SDK. Con el AVD *Manager* (Figura 2. 14) se pueden ver, crear y eliminar AVDs. Crear y actualizar proyectos Android. Actualizar el Android SDK con nuevas versiones, documentación, etc.
emulator **(Emulador Android)**	Emulador de dispositivo móvil (Figura 2. 13), dispositivo móvil virtual que se ejecuta en el ordenador permitiendo diseñar, desarrollar y probar aplicaciones en un entorno de ejecución sin necesidad de utilizar un dispositivo físico.
ddms	DDMS (*Dalvik Debug Monitor Server*) permite la depuración de aplicaciones Android (Figura 2. 15).
hierarchyviewer **(Hierarchy Viewer)**	Herramienta que permite la depuración y optimización de la interfaz de usuario de las aplicaciones Android. Proporciona una representación visual de la estructura de *layouts*.
sqlite3	Permite el acceso a los ficheros de datos de SQLite creados y usados por las aplicaciones Android.
traceview	Proporciona un visor gráfico para guardar los registros de la ejecución de la aplicación.
mksdcard	Permite crear una imagen que se usa con el emulador y que simula la presencia de un dispositivo de almacenamiento externo.
dmtracedump	Genera diagramas de pila de llamadas, a partir de los archivos de registro de seguimiento.
Draw 9-patch	Permite crear fácilmente un gráfico *NinePatch* usando el editor WYSIWYG (*What You See Is What You Get*) (ver Figura 2. 16).
layoutopt	Permite analizar rápidamente los *layouts* de la aplicación con el fin de optimizarlos.
Monkey	Herramienta que se ejecuta en el emulador o dispositivo y genera secuencias pseudoaleatorias de eventos de usuario.
monkeyrunner	Proporciona una API para escribir programas que controlan un dispositivo o emulador Android desde el exterior del código Android.
ProGuard	Reduce y optimiza el código gracias a la eliminación de código no utilizado.

Tabla 2. 6: Herramientas de desarrollo: SDK *tools*

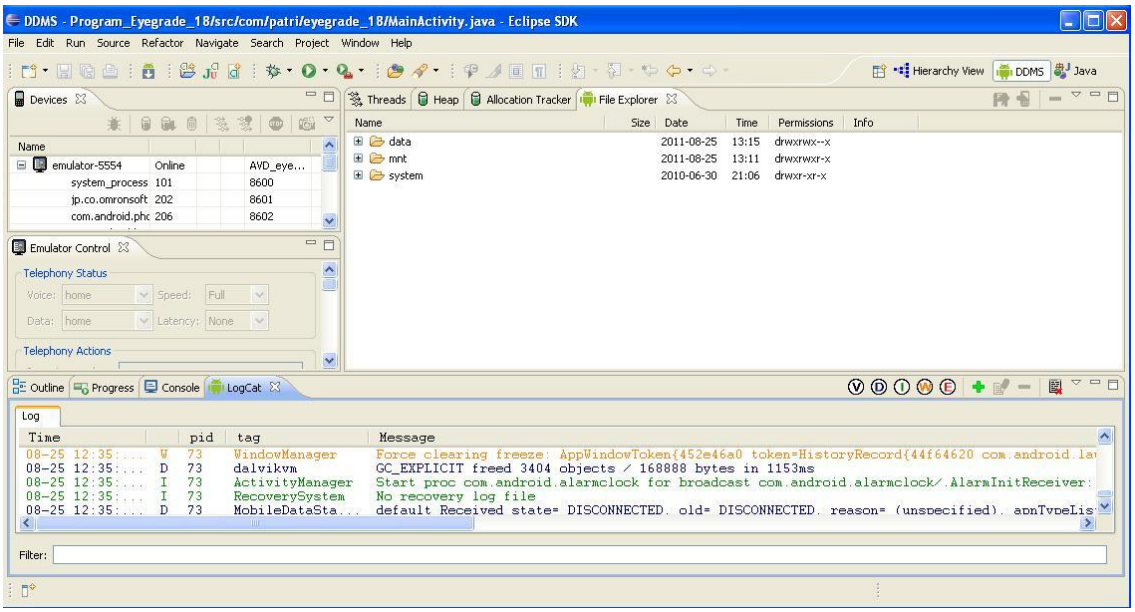


Figura 2. 15: Vista DDMS

Herramienta	Descripción
`adb`	ADB (*Android Debug Bridge*) es una versátil herramienta que permite administrar el estado del emulador o del dispositivo. Usado también para instalar una aplicación Android (un fichero `.apk`) en el dispositivo.
`aidl`	AIDL (*Android Interface Definition Language*) permite definir la interfaz de programación para la comunicación entre cliente y servidor mediante la IPC (*Inter-Process Communication*).
`aapt`	AAPT (*Android Asset Packaging Tool*) construye los archivos `.apk`.
`dx`	Herramienta que convierte los archivos *bytecode* de Java (`.class`) en archivos *bytecode* de Android (`.dex`) (Ver apartado 2.3.1.5).

Tabla 2. 7: Herramientas de desarrollo: *Platform tolos*

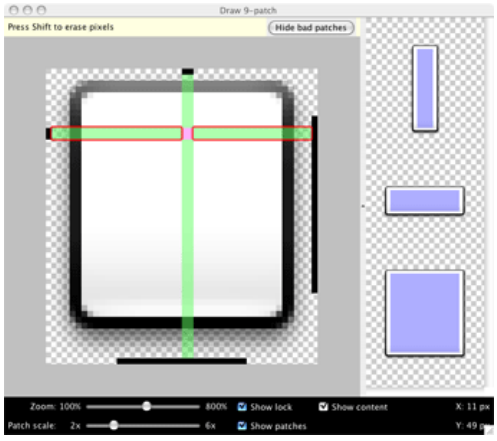


Figura 2. 16: Herramienta `Draw 9-patch` [14]

2.4 Lenguaje de programación Python

Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Combina potencia con una sintaxis muy clara. Es un lenguaje de programación fácil de aprender a la vez que portable, usado para el desarrollo de aplicaciones en múltiples plataformas [19].

2.4.1 Historia

Python fue creado, como sucesor del lenguaje llamado ABC, a principios de los años noventa por Guido Van Rossum, cuando éste trabajaba en el *Stichting Mathematisch Centrum* (CWI, *Centrum Wiskunde & Informatica*) de los Países Bajos. Guido es el principal autor, aunque en este lenguaje de programación se incluyen contribuciones de otros colaboradores. En 1995 Guido continuó su trabajo sobre Python en el CNRI (*Corporation for National Research Initiatives*) de Reston, Virginia, donde creó varias versiones del *software*. En el año 2000, Guido y el equipo de desarrollo de Python se trasladan a BeOpen.com, formando el equipo BeOpen PythonLabs.

Después de lanzar la única versión de Python con BeOpen.com, Guido y su equipo de desarrolladores se unieron a la empresa *Digital Creations*. A partir de entonces, la propiedad intelectual pertenece a la PSF (*Python Software Foundation*), una organización sin ánimo de lucro, creada específicamente para proteger la libertad de Python como lenguaje de código abierto [20].

2.4.2 Principales características

El lenguaje de programación Python tiene múltiples características, y entre ellas las más importantes son:

Sintaxis legible: Python tiene una sintaxis simple, convencional y muy clara, lo que facilita en gran medida su lectura. Las sentencias de Python son similares a los algoritmos de pseudocódigo y sus expresiones usan la notación convencional del álgebra [21].

Semántica segura: al tener una semántica de seguridad, las expresiones que incumplan las reglas del lenguaje producirán un mensaje de error [21].

Programación orientada a objetos: es uno de los estilo de programación que soporta Python de forma intuitiva. Los programas se basan en clases y objetos que representan los conceptos necesarios de cada aplicación.

Lenguaje de programación interpretado: se ejecuta directamente desde el código fuente escrito por los desarrolladores (en contraposición a los lenguajes compilados que deben traducirse a código máquina antes de ser ejecutados). La ejecución de los lenguajes interpretados es más lenta al tener que hacer la traducción sobre la marcha, pero el tiempo de desarrollo y depuración es más rápido al no tener que esperar al compilador [22].

Lenguaje extensible, modularidad completa: Python permite dividir el código en distintos módulos reutilizables para otros programas, simplemente importándolos. Es un lenguaje ampliable con módulos de extensión de C o C++ que lo hacen adecuado para múltiples aplicaciones y sistemas operativos.

Tipos de datos de alto nivel: lenguaje que soporta estructura de datos tan complejas como puedan ser listas o diccionarios (estructuras que pueden almacenar otros tipos de datos en su interior) que incluyen una gran funcionalidad [22].

Biblioteca estándar: lenguaje poseedor de extensas bibliotecas estándar y módulos de terceros, para prácticamente todas las tareas. La biblioteca de Python, *The Python Standard Library*, ofrece una gran cantidad de funciones que cubren desde el procesamiento asíncrono hasta los archivos comprimidos [19].

De propósito general: el lenguaje incluye recursos para aplicaciones actuales, como puedan ser sistemas multimedia o computación de redes [21].

Lenguaje de tipos dinámicos: no se utilizan declaraciones explicitas de tipos de datos. Los tipos se descubren en el momento de la ejecución. El programa deduce el tipo de una variable cuando se le asigna un valor por primera vez [20].

Lenguaje fuertemente tipado: una vez que se le ha dado un tipo de dato a una variable, éste permanece. En el lenguaje Python los tipos se mantienen siempre, no es posible tratar una variable como si fuese de otro tipo sin convertirla explícitamente [20].

Interactivo: se incluye un modo interactivo en el que las instrucciones y declaraciones pueden introducirse una a una en el intérprete de comandos. Esto facilita al programador probar el código experimental viendo el resultado inmediatamente. Se puede probar porciones de código en el modo interactivo, antes de integrarlos en segmentos mayores, (guardados en ficheros) que serán cargados y ejecutados en módulos o aplicaciones independientes [21].

Lenguaje de programación multiparadigma: soporta más de un estilo o filosofía de programación. Beneficioso para las aplicaciones que tienen un planteamiento flexible con respecto a la programación. Python incluye herramientas para la programación estructurada, funcional y orientada a objetos [22].

Lenguaje de código abierto y de uso generalizado: se puede descargar Python en una amplia variedad de dispositivos fácilmente. Existe una gran comunidad de usuarios y desarrolladores de Python, que contribuyen en el mantenimiento de una documentación muy completa [21].

Multiplataforma: el intérprete de Python está disponible en multitud de plataformas, es decir, es fácil de ejecutar en muchos sistemas operativos (entre ellos Windows, Linux y Mac). Si no se utilizan bibliotecas especificas de cada plataforma, el programa Python podrá ejecutarse en esos distintos SO, sin necesidad de hacer modificaciones.

2.4.3 Elementos del lenguaje

Algunas de las peculiaridades del lenguaje de programación Python se encuentran descritas en la Tabla 2. 8, mientras que en la Tabla 2. 9 se representan algunos de los distintos tipos de datos existentes en Python.

Elemento			Descripción
`and`	`or`	`not`	Operadores lógicos con palabras y no con símbolos para facilitar su lectura.
	tabulador		Uso del tabulador para mostrar estructura de bloques.
	`#`		Indicación de un comentario hasta el final de la línea.
	`def`		Palabra clave usada para definir una función.
	`import`		Palabra clave para agregar un nuevo módulo.

Tabla 2. 8: Características del lenguaje Python

Python se utiliza con éxito en miles de aplicaciones empresariales del mundo real en todos los países, incluyendo sistemas muy grandes y de importancia crítica. Alguno ejemplos del uso de Python son YouTube, Industrial Light & Magic y Google.

Python no es una excepción, y como todos los lenguajes de programación, no es perfecto. Python es un lenguaje de alto nivel, y no es por tanto adecuado para programación de bajo nivel (como la programación de *drivers* o *kernels*) ni para la programación de aplicaciones que requieran alta capacidad de cómputo.

Dato	Tipo	Declaración/ejemplo	Descripción
`str`	*String*	comillas o dobles comillas. **'python'** o**"python"**	Cadena de caracteres.
`unicode`	*String*	método **unicode('python')** ó **u' python'**	Cadena de caracteres *unicode.*
`int`	Entero	**48**	Número fijo.
`long`	Entero	letra L al final del número **48L**	Número entero con mayor precisión.
`float`	Coma flotante	un punto para señalar la parte decimal **48.4562**	Numero en coma flotante.
`bool`	Booleano	True o False	Valor booleano.
`list`	Secuencia (lista)	Entre corchetes **[2, False, 'python', 3.4]**	El contenido **si** puede variar en tiempo de ejecución y puede ser de diferentes tipos de datos.
`tuple`	Secuencia (tupla)	Entre paréntesis (2, False, 'python', 3.4)	El contenido **no** puede variar en tiempo de ejecución y puede ser de diferentes tipos de datos.
`set`	Conjunto	Con la instrucción set() **set([2, False, 'python', 3.4])**	El contenido **si** puede variar en tiempo de ejecución y puede ser de diferentes tipos de datos. No hay orden ni elementos duplicados.
`frozenset`	Conjunto	Con la instrucción frozenset() **frozenset([2, False, 'python', 3.4])**	El contenido **no** puede variar en tiempo de ejecución y puede ser de diferentes tipos de datos. No hay orden ni elementos duplicados.
`dict`	*Mapping* (diccionario)	Entre llaves, par clave-valor **{'clave0': 2, 'clave1': False}**	Conjunto de pares formados por la clave y su valor.

Tabla 2. 9: Tipos de datos del lenguaje Python

2.5 Scripting Layer for Android (SL4A)

SL4A (*Scripting Layer for Android*) es una aplicación, una herramienta, dirigida a todos los desarrolladores de aplicaciones Android, que buscan una manera de escribir programas con cualquiera de los lenguajes soportados y diferentes a Java. Los lenguajes que soporta en la actualidad SL4A son: Beanshell, JRuby, Lua, Perl, PHP, Python y Rhino.

Se proporciona una consola interactiva en el dispositivo Android, donde poder escribir una línea de código y comprobar el resultado inmediatamente. Se permite incluso en muchos casos, la reutilización de código que se haya escrito para un entorno de escritorio.

SL4A hace posible la escritura de código para dispositivos Android en otros lenguajes a parte de Java, y hacerlo de forma más interactiva [23].

SL4A, originalmente llamado ASE (*Android Scripting Environment*), fue creado por Damon Kohler en 2009 y está alojado en Google Code[3] como un proyecto de código abierto. La versión disponible más actualizada que existe hoy en día es la llamada r4 [24].

2.5.1 Arquitectura

En su nivel más bajo, SL4A es esencialmente un motor y entorno de ejecución de *scripts*, lo que quiere decir que como aplicación alberga diferentes interpretes, cada uno de los cuales procesa un lenguaje específico.

El repositorio del código fuente de SL4A está estructurado según una jerarquía de árbol con cada lenguaje [24]. La transferencia de este código fuente a la plataforma Android, se consigue con compilación cruzada para una arquitectura ARM (*Advanced RISC Machine*) a través del NDK (*Android Native Development*). Para evitar cambios en el código fuente, éste se mantendrá en su estado puro, lo que conlleva que los lenguajes de *script* no tengan conocimiento alguno sobre la plataforma Android. Una vez que el SL4A lance un intérprete concreto, el módulo característico que da acceso a la API de Android se cargará como una biblioteca, y a partir de entonces los *scripts* serán interpretados línea a línea.

La arquitectura básica de SL4A es similar a la de un entorno de computación distribuida. En la Figura 2. 18 se muestra gráficamente el flujo de ejecución que se produce con el lanzamiento de SL4A y posterior ejecución del programa *Hello World* típico, en este ejemplo, escrito en Python, en el *script* `hello.py`. Cada *script* SL4A debe provenir de un fichero externo (fichero `android.py` para Python por ejemplo) que será el que defina una serie de funciones *proxy* necesarias para comunicarse con la API de Android.

La comunicación real entre SL4A y el sistema operativo Android subyacente se basa, en el protocolo de llamada a procedimiento remoto, RPC (*Remote Procedure Call*) y en el formato de datos JSON (*JavaScript Object Notation*). El mecanismo RPC se utiliza normalmente en las arquitecturas distribuidas en las que hay una transferencia de información entre cliente y servidor. En el caso de SL4A, el SO Android es el servidor y el fichero SL4A el cliente [23]. El servidor RPC (implementado como una aplicación Java de Android) tendrá acceso directo a la API de Android y se comportará como un *proxy* remoto mediante el patrón de diseño '*Java Android API* Facade' (Figura 2. 17) que encapsula y proporciona el acceso a las APIs seleccionadas de Android [25].

[3] Google Code: http://code.google.com *Web* oficial de Google para desarrolladores de proyectos.

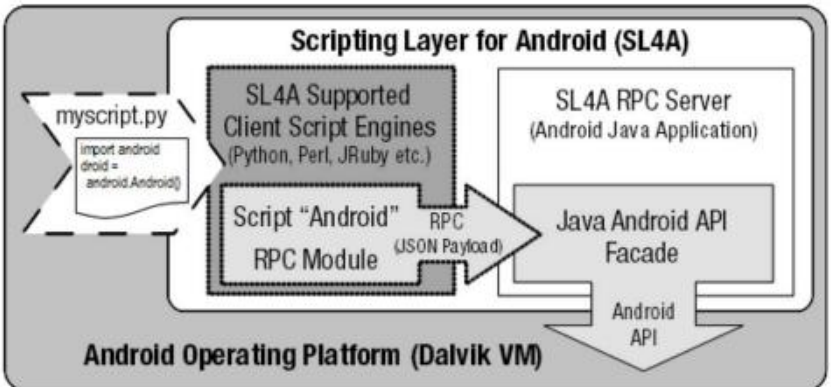


Figura 2. 17: Arquitectura SL4A [25]

El tipo de comunicación que existe en la arquitectura SL4A (Figura 2. 17) consigue aumentar el nivel de separación entre ambos, y así, evitar cualquier posible *script* malintencionado que pudiese intentar hacer algo perjudicial para el sistema [23].

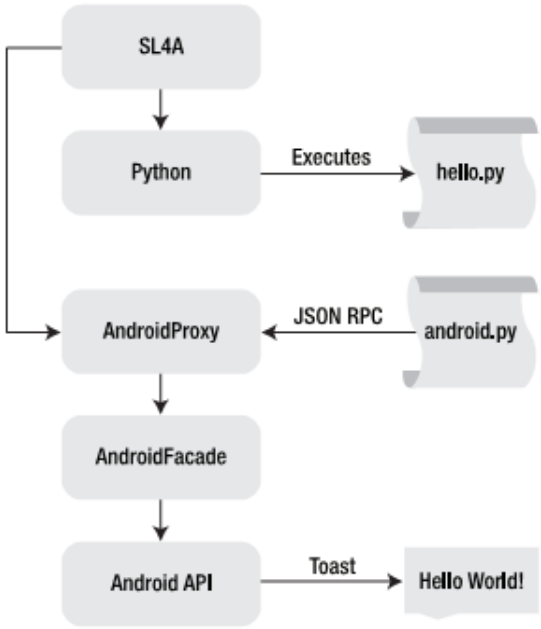


Figura 2. 18: Diagrama de flujo de ejecución de SL4A [23]

En resumen, la arquitectura entera de SL4A y su flujo de ejecución, se basan en un motor de ejecución de *scripts* que se ejecuta en un proceso autónomo, accediendo a la API de Android mediante una comunicación bidireccional JSON-RPC a través del API *facade*. Este último es llevado por un proceso aparte e implementado como un servidor de aplicaciones Android estándar basado en Java. Dicho servidor tiene acceso completo a la API de la plataforma Android actuando por tanto, como un *proxy* remoto de la API para el intérprete de *scripts*. Cada lenguaje que soporta SL4A posee un módulo '*wrapper*' (contenedor, envolvente) que es el que proporciona el 'objeto Android' para el acceso a la API de Android [25].

2.6 Aplicación original EYEGRADE: Visión artificial aplicada al ámbito educativo

La base de este proyecto fin de carrera es EyeGrade[4], una aplicación implementada en Python para la corrección automática de exámenes de respuestas múltiples mediante cámara *web*.

La versión del programa original que se ha utilizado (a partir de la cual, se ha generado la aplicación Android) ha sido la versión 0.1.6.1, que es la versión más reciente disponible en el momento de comenzar la migración. Las posteriores versiones y mejoras de dicha aplicación para PC (*Personal Computer*) no se han implementado en el sistema Android.

2.6.1 Funcionamiento de la aplicación

El usuario introduce por la línea de comandos del ordenador una serie de datos de entrada, que el programa recibe y procesa para poder realizar correctamente su función. Gracias a estos parámetros introducidos, los exámenes que se pueden analizar son de carácter dinámico, es decir, no todos los exámenes tienen que tener las mismas características. De las diferentes opciones que se pueden dar en la configuración de los exámenes, las dos más relevantes se basan en el número de preguntas y en el número de respuestas posibles a cada pregunta.

Cada vez que se ejecuta la aplicación, se configuran dichos parámetros, pudiendo tener un número de preguntas y número de opciones por preguntas independientes unos exámenes de otros, según la configuración inicial que se realice. Por ejemplo, se pueden analizar tanto exámenes que consten de 20 preguntas y cada una de ellas con 4 opciones, como exámenes con 30 preguntas de 2 opciones cada una o 15 preguntas con 3 opciones.

La existencia de diferentes modelos dentro de un mismo tipo de examen es otra cualidad de este programa. En otras palabras, un examen que tenga unas ciertas características (número de preguntas y respuestas) puede tener distintos modelos, es decir, distintas permutaciones en el orden de las preguntas y en el orden de las soluciones dentro de cada pregunta. Este hecho se presenta habitualmente para evitar posibles copias en asignaturas donde un gran número de alumnos realizan un mismo examen. De esta manera, los estudiantes no pueden fijarse en la casilla marcada por ningún otro alumno. La aplicación EyeGrade, con la configuración apropiada, puede corregir todos los exámenes de dicha asignatura sin necesidad de reiniciarse. El fichero de configuración posee las soluciones de todos los diferentes modelos y, según sea el modelo detectado por la cámara, así procederá al calificarlo.

Otra posibilidad ofrecida por el programa EyeGrade para PC es la selección de la cámara *web*[5], pudiendo elegir entre las múltiples cámaras que tenga conectadas al ordenador.

Una vez que se realiza la captura de la imagen de un examen y su posterior procesado, se presenta en la pantalla los resultados de dicho análisis, dejando la imagen fija del examen con la información superpuesta (ver Figura 2. 19). La aplicación, entonces, permite guardar en

[4] Página oficial del programa EyeGrade: http://www.it.uc3m.es/jaf/eyegrade/
[5] La selección de la cámara es una funcionalidad no portada a la aplicación para dispositivos Android al tratarse de terminales con una sola cámara.

el ordenador las imágenes obtenidas como resultado del procesamiento del examen, así como corregir fallos que el programa pueda cometer en la corrección.

La aplicación, sin embargo, no consta únicamente de las funciones correspondientes a la corrección del examen a través de una visión artificial, sino que también permite maquetar los exámenes. A partir de preguntas escritas en XML genera el enunciado de los distintos modelos del examen, así como el fichero de configuración necesario para su corrección. La maquetación del examen se basa en el uso de Latex[6].

Además de guardar las imágenes con todos los detalles añadidos por el programa (respuestas correctas, respuestas incorrectas, identificador del estudiante, etc.), se guardan también las imágenes del examen sin ningún dato superpuesto, para poder realizar, si se desea, un futuro análisis o depuración de la imagen del examen. Esto implica que el programa tiene un modo de funcionamiento, en el que no utiliza la cámara para capturar imágenes de los exámenes a procesar, sino que acepta como parámetros de entrada, imágenes de exámenes ya capturadas.

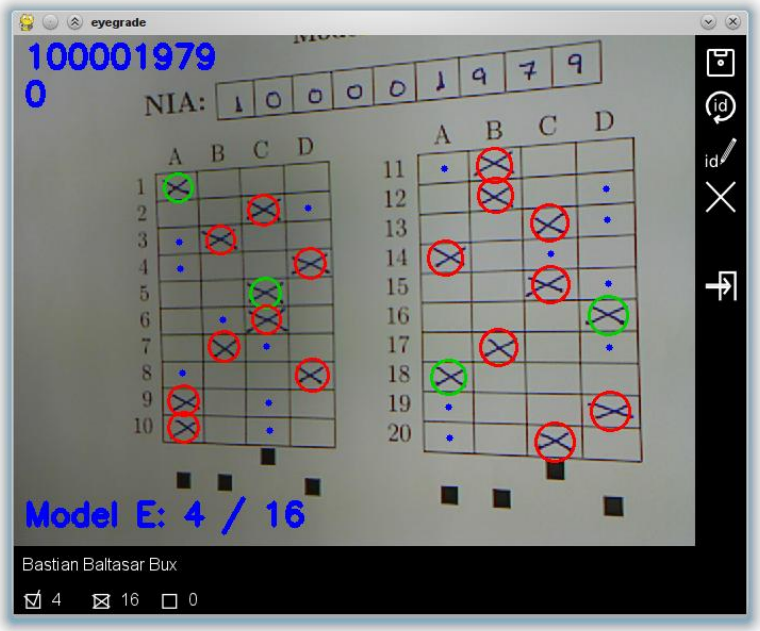


Figura 2. 19: Captura de pantalla del programa EyeGrade para PC [26]

[6] Fichero LaTeX: archivo fuente que permite ser exportado a numerosos formatos como PDF (*Portable Document Format*), RTF (*Rich Text Format*), HTML (*HyperText Markup Language*) ó Postscript entre otros.

En la Tabla 2. 10 se describen los principales parámetros de la parte, dentro del programa, relacionada con la creación de exámenes. Algunos de estos parámetros (por ejemplo, el nombre de la asignatura o la fecha de examen) pueden también introducirse mediante el fichero XML que contiene las preguntas.

Nombre	Parámetro de entrada/salida	Condición	Descripción
Output File	entrada	opcional	Parámetro introducido por el usuario y utilizado para dar nombre a los archivos de salida.
Exam File	entrada	obligatorio[7]	Fichero XML que contiene las preguntas del examen.
Num Questions	entrada	obligatorio[7]	Número de preguntas totales del examen.
Num Choices	entrada	obligatorio[7]	Número de opciones posibles de respuestas de cada pregunta del examen.
Num Tables	entrada	opcional	Numero de tablas gráficas en las que se repartirán las preguntas.
Date	entrada	opcional	Fecha del examen.
Subject	entrada	opcional	Nombre de la asignatura del examen.
Degree	entrada	opcional	Carrera en la que se imparte la asignatura.
Models	entrada	opcional	Distintos tipos/modelos de exámenes.
Duration	entrada	opcional	Tiempo de duración del examen.
Do Not Shuffle	entrada	opcional	Booleano que permite mantener la permutación ya existente en el fichero de configuración del examen.
Template Filename	entrada	obligatorio	Fichero Latex usado como plantilla para la creación del examen.
output_file.tex	salida	siempre se obtendrá al menos uno	Varios ficheros Latex de salida para crear un examen por cada modelo que se haya especificado.
config_filename.eye	salida	Dependiente de la existencia del parámetro *'Output File'*	De igual nombre que '*Output File*'. Fichero de configuración necesario para la corrección futura del examen por parte de EyeGrade (parámetro de entrada '*Exam Data File*' en la Tabla 2. 11).

Tabla 2. 10: Parámetros de EyeGrade: creación de exámenes

[7]Es obligatorio la introducción de bien '*Exam File*' o bien el conjunto de '*Num Questions*' y '*Num Choices*'. Siendo obligatorio una de las dos opciones, son a la vez mutuamente excluyentes, no pudiendo introducirse los tres parámetros juntos.

En la siguiente tabla, la Tabla 2. 11, se presentan, al igual que se ha hecho con el apartado anterior, los parámetros de entrada y salida de la pieza fundamental de la aplicación, es decir, del visionado de exámenes y corrección de éstos.

Nombre	Parámetro de entrada/salida	Condición	Descripción
Exam Data File	entrada	obligatorio	Fichero de configuración `.eye` obtenido en la sección de la aplicación dedicada a la creación de exámenes, Tabla 2. 10. Fichero necesario para poder identificar y corregir el examen en cuestión.
Start-id	entrada	opcional	Entero con el que comenzar a identificar las capturas.
Output Dir	entrada	opcional	Directorio de salida donde guardar los archivos obtenidos.
Debug	entrada	opcional	Booleano que indica si habilitar o no, el modo de depuración.
Save_Stats	entrada	opcional	Booleano usado para guardar o no, parámetros de rendimiento en el archivo de `answers.csv`.
Id List File	entrada	opcional	Fichero `.csv` que contiene la lista con los identificadores de los estudiantes.
Capture File	entrada	opcional	Ficheros `.png` ('RAW' o 'PROC')[8] usados para la depuración. Ficheros utilizados en lugar del uso de la cámara.
Fixed Hough	entrada	opcional	Entero correspondiente a un parámetro de control para la visión artificial y autoajustable en caso de no ser introducido
Adjust	entrada	opcional	Booleano usado para activar la captura de imágenes manualmente.
exam.png	salida	opcional	Imagen de salida del examen procesado que se almacenará si el usuario lo desea.
exam.png-proc	salida	opcional	Imagen 'proc' de salida del examen que se almacenará si el usuario lo desea para una posible futura depuración.
exam.png-raw	salida	opcional	Imagen 'raw' de salida del examen que se almacenará si el usuario lo desea para una posible futura depuración.
answers.csv	entrada-salida	opcional	Fichero `.csv` de salida y seleccionable a la entrada con las respuestas y resultados de los alumnos que se almacenará si el usuario lo desea.

Tabla 2. 11: Parámetros de EyeGrade: corrección de exámenes

[8] La imagen RAW es una imagen RGB (*Red, Green, Blue*) y la imagen PROC es una imagen monocromática obtenida como resultado del paso (1) del apartado 2.6.2.

2.6.2 Características del sistema de visión artificial

Los criterios seguidos en el diseño del sistema de visión artificial del programa EyeGrade son:

- La precisión en la detección es más importante que el tiempo de detección.
- Se puede capturar un flujo continuo de imágenes.
- Los usuarios deben ser capaces de revisar y corregir, si fuera necesario, las decisiones automáticas del sistema.

Siguiendo estos criterios y una vez que se ha capturado una imagen, el sistema realiza los siguientes pasos:

(1) **Procesamiento inicial de la imagen**: la imagen en color se transforma en una imagen monocromática. En primer lugar, los tres canales RGB se transforman en un canal de escala de grises, y posteriormente se aplica un algoritmo de umbral adaptativo con el fin de transformar la imagen a blanco y negro.

(2) **Detección de líneas**: las líneas rectas se detectan en la imagen mediante el uso de la transformada de Hough [27], representando cada línea detectada por su mínima distancia ρ al origen de coordenadas y el ángulo θ del vector que va desde el origen hasta el punto más cercano de la línea.

(3) **Detección de la tabla de respuestas**: la geometría de la o las tablas donde se han escrito las respuestas se detecta a través de la intersección de las líneas obtenidas en el paso anterior. Se clasifican las líneas en dos grupos (verticales y horizontales), se comprueba que se haya detectado el número correcto de líneas y se calculan las coordenadas de las esquinas de cada casilla de las tablas.

(4) **Toma de decisiones**: el programa analiza la parte de la imagen correspondiente al interior de cada casilla y decide si ésta ha sido marcada o no. El sistema espera que se marquen las celdas con una cruz (dos líneas que unen las esquinas opuestas de la casilla).

(5) **Detección del modelo**: el sistema admite la codificación del modelo de examen en la hoja de respuesta, detectándolo en la imagen capturada. Un conjunto de cuadrados negros impresos en la parte inferior codifican, con un alto grado de redundancia, un número binario que representa el identificador del modelo.

(6) **Detección del identificador del estudiante**: el sistema aplica las técnicas OCR (*Optical Character Recognition*) para detectar los dígitos escritos a mano correspondientes con los identificadores de los estudiantes.

Capítulo 3

Alternativas de diseño

3.1 Introducción

En este capítulo se va a realizar la exposición de las alternativas del diseño llevado a cabo en este proyecto fin de carrera. Este diseño se ha realizado al migrar un programa creado para PC (aplicación EyeGrade explicada en el apartado 2.6) a un dispositivo móvil basado en tecnología Android.

Se va a presentar la idea original que se pensaba llevar a cabo, y las razones por las que no se pudo implementar dicho diseño, pasando posteriormente, a presentar el diseño final que se ha realizado, indicando sus características y rasgos más importantes. Se mostrará el diseño implementado sin entrar en detalles concretos sobre el código creado (los detalles de la implementación se encuentran en el apartado 5.3). Más bien, se tratarán las ideas y soluciones requeridas para lograr el objetivo buscado, que no es otro que la migración de la aplicación EyeGrade.

El programa inicial está escrito en lenguaje Python (explicado en el apartado 2.4), mientras que el desarrollo de las aplicaciones de la plataforma Android se realiza a través del lenguaje de programación Java. Por lo tanto, la migración de la aplicación EyeGrade presenta una discrepancia entre los lenguajes de programación Python y Java.

Por consiguiente, la primera incompatibilidad de la migración se presenta en los lenguajes de programación, el programa EyeGrade necesita Python, mientras que los dispositivos móviles basados en Android necesitan Java. Esta diferencia entre requisitos presenta dos posibles soluciones iniciales, usar el código Python en Android o convertir el código original a Java. Es decir, o se crea una aplicación para Android desarrollada en código Python, o se escribe de nuevo el código del programa EyeGrade en Java, para que se pueda ejecutar en la plataforma Android.

La idea que se ha tenido desde un principio ha sido la primera: el programa original escrito en Python es el que se tenía que migrar a la plataforma Android. Por tanto, la segunda alternativa fue descartada desde el comienzo.

Una vez que se ha decidido optar por el uso del código original Python, se necesita la herramienta SL4A (ver apartado 2.5) para llevar a cabo dicha tarea. Partiendo por tanto de esta herramienta (que permite realizar aplicaciones Android escritas en lenguajes distintos de Java, en este caso Python), se va a proceder a presentar las distintas alternativas de diseño.

3.2 Alternativa inicial

Inicialmente, se intentó encontrar la manera de utilizar el código Python en su totalidad, creando a través de la herramienta SL4A, una aplicación Android basada en código Python.

Esta alternativa inicial se basa en la reutilización del código original, adaptándolo a las nuevas circunstancias en algunos aspectos, como puedan ser la entrada de datos o la interfaz gráfica de usuario.

3.2.1 Herramienta SL4A

Para realizar este proyecto, se necesita algún mecanismo que permita realizar una aplicación Android en código Python. La búsqueda de la herramienta que admitiese esta acción, así como su estudio y posterior aplicación, eran, en inicio, la parte más importante a realizar. Encontrar la posibilidad de ejecutar código Python, sobre un dispositivo Android es algo que se ha logrado mediante la herramienta *Scripting Layer for Android* (ver apartado 2.5).

Dicha herramienta es bastante actual y novedosa, características siempre positivas para encontrar información en Internet, pero también estas características conllevan unas desventajas como que la documentación aún no esté muy bien estructurada.

Una vez que se maneja la aplicación SL4A y se ha decidido que es la única solución de diseño para realizar la migración, el punto clave es desarrollar un proyecto Android con dicha herramienta. Es decir, no solo ejecutar un *script* Python en un dispositivo Android, sino crear una aplicación entera, un proyecto con todos sus elementos básicos.

3.2.2 Comienzo de la migración

La adaptación de la aplicación se ideó siguiendo la línea de ejecución del programa, comenzando de esta manera por la introducción de los parámetros de configuración y acabando con la obtención de los datos de salida.

Es esta la razón de que se hubiese comenzado con la creación de los menús de entrada de datos, creando en código Python, la GUI imprescindible para obtener del usuario los parámetros de configuración. Con esta sección de código se consigue la información que, en el programa original para ordenador, se adquiría por la línea de comandos.

Cabe señalar que este hecho, el cambio de pasar información por línea de comandos a hacerlo mediante menús gráficos, es una adaptación del código original a la nueva plataforma, lo que entra dentro de los planes del diseño inicial. No se puede utilizar el código original por tratarse de una plataforma diferente, donde la interacción con el usuario se realiza de forma distinta.

Por tanto, aunque no se ha podido reutilizar el código Python (se ha tenido que adaptar a la UI de un dispositivo móvil), la alternativa inicial sigue su curso. Una vez terminados todos los menús de inicio (con todas las implicaciones que estos conllevan, como por ejemplo, la búsqueda de ficheros dentro del dispositivo móvil o la verificación de haber introducido los parámetros mínimos indispensables), se pasa a la realización de la primera sección de la aplicación, la creación de ficheros de examen.

La implementación de este conjunto se ha realizado dentro de lo ideado en un principio y en base a lo estipulado en el código original, salvo partes concretas que usaban el intérprete de comandos. El único inconveniente encontrado ha sido la falta de información de los errores producidos en Android con lenguaje Python.

3.2.3 Problemas encontrados

Cuando se comienza a efectuar la migración del código Python de la segunda parte de la aplicación, la corrección visual de los exámenes, es el momento donde surgen los problemas de esta alternativa inicial del diseño.

Es entonces cuando se ven los inconvenientes que presenta esta opción elegida, y donde, a pesar de intentar solventarlos, se tienen que debatir otras vías alternativas de implementación para lograr llevar a cabo el objetivo final.

A continuación se van a plantear los problemas encontrados, problemas que son los causantes de haber tenido que desechar esta alternativa de diseño y haber implementado la que se expone en el apartado 3.3.

En la Tabla 3. 1 se exponen dichos inconvenientes de forma esquemática. Los problemas presentados brevemente en dicha tabla serán explicados posteriormente con mayor detenimiento.

Problema encontrado	Descripción
Biblioteca OpenCV	No existencia de la biblioteca OpenCV (*Open Source Computer Vision*) para Python sobre Android.
Comunicación entre lenguajes	Necesidad de implementar una comunicación entre los lenguajes Python y Java.
Interfaz gráfica de usuario	Realización de la GUI de la aplicación, basada en el programa original.
Captura de imágenes	Adaptación a la cámara del dispositivo Android para realizar la toma de imágenes.

Tabla 3. 1: Problemas encontrados en la alternativa del diseño inicial

- **Biblioteca OpenCV:**

La aplicación EyeGrade, en su sección dedicada a la visión artificial (funcionalidad principal), se basa en el uso de la biblioteca OpenCV, dicha biblioteca es imprescindible para la implementación de la visión artificial en esta aplicación. Sin embargo, la disponibilidad de la biblioteca OpenCV para Python en Android no es factible. La imposibilidad de disponer de dicha biblioteca en Python para Android hace que no sea posible mantener el código original.

Este inconveniente plantea la posibilidad de desarrollar dos soluciones diferentes, la transmisión de la imagen capturada a un servidor y su posterior recepción de las imágenes procesadas, o la búsqueda de la biblioteca OpenCV para Java en Android. Se ha optado por la segunda de ellas, intentando conseguir una aplicación autónoma, manteniendo la arquitectura

de la aplicación en los mínimos dispositivos posibles (ver apartado 4.3) sin hacer uso de un nuevo módulo (sin que un servidor sea un requisito para la aplicación EyeGrade).

- **Comunicación entre lenguajes:**

El uso de la biblioteca en Java hace necesario un cambio de la perspectiva en el diseño original, ya que no se va a poder realizar el código de la aplicación completamente en Python. Esto conlleva tener que implementar una comunicación entre la parte escrita en lenguaje Python y la parte que se desarrolle en Java.

Esta comunicación entre ambos lenguajes, no ideada en un principio, es un nuevo punto de estudio a tener en cuenta para la nueva alternativa de diseño. Se ha de encontrar la forma de realizar tal comunicación, estudiando los requisitos y las limitaciones que conlleve.

- **Interfaz gráfica de usuario:**

Una vez creada esta vía de comunicación entre ambos lenguajes de programación, se plantea el dilema de la implementación de la interfaz gráfica de usuario encargada de mostrar la visualización realizada del examen, aspecto también importante a tener en cuenta en el diseño de la migración. En este caso, aparte de realizarlo en Java, se tiene la opción de implementarlo en Python mediante código HTML, es decir, crear interfaces gráficas en Android sobre SL4A.

La decisión tomada con respecto al diseño a realizar (GUI de la captura de imágenes), es la primera opción planteada, es decir, uso de *layout* con ficheros XML y código Java. El motivo de tal decisión ha sido la imposibilidad de mantener de forma fija en la pantalla, una interfaz realizada en Python si es necesario realizar también operaciones en Java (como es el caso de EyeGrade al necesitar OpenCV).

- **Captura de imágenes:**

El último escollo reseñable de esta primera alternativa fallida es el referente a la captura de imágenes. La toma de fotografías por parte del dispositivo móvil Android y su posterior manipulación se ha tenido que realizar en Java con motivo de la adaptación de la biblioteca OpenCV para Java sobre la plataforma Android.

Partiendo por tanto del lenguaje de programación Java y teniendo disponible la biblioteca OpenCV, se intentó implementar esta funcionalidad siguiendo los pasos del programa original, que realiza la captura de imágenes mediante funciones de dicha biblioteca. Sin embargo, no se pudo llevar a cabo con éxito esta idea ya que OpenCV no da soporte a la captura de imágenes con la cámara de Android.

Por consiguiente, la toma de imágenes se ha ejecutado a través de las herramientas propias de la cámara de Android.

3.3 Alternativa implementada

La alternativa del diseño que se ha llevado a cabo finalmente, ha sido un mezcla entre las dos opciones que se barajaban inicialmente, todo en código Python o todo en código Java. El diseño elegido es una mezcla porque tiene ambos lenguajes unidos, realizando una parte de la aplicación con el lenguaje original y otra sección del código con el lenguaje propio de Android, evitando así la reescritura en Java del código Python que si ha sido posible migrar a la nueva plataforma.

La parte del programa implementada en Python es la parte comentada en la sección anterior (apartado 3.2), correspondiente con la configuración de los parámetros de entrada y con toda la primera parte de la aplicación EyeGrade, creación de exámenes.

La funcionalidad más importante (visión artificial de exámenes), sin embargo, será la que comparta ambos lenguajes de programación, comenzando con Python y pasando a lenguaje Java cuando comience el procesado de imágenes.

El hecho de escribir parte de la aplicación en Python y parte en Java, implica una serie de requisitos que se deben cumplir para validar el diseño seleccionado. Las principales características de esta alternativa implementada se presentan a continuación.

3.3.1 Conexión entre los lenguajes Python y Java

El diseño elegido requiere la comunicación entre ambos lenguajes. Hecho que, aunque no es sencillo, es realizable gracias a una de las últimas actualizaciones de la herramienta *Scripting Layer for Android*. La conexión creada desde Python hacia una *activity* de Android refleja el potencial de la aplicación SL4A, pero también su fragilidad para implementar ciertas acciones (debido a estar en pleno proceso de desarrollo). Esto es así, puesto que con la versión con la que se inició este proyecto, no era posible tal diseño, mientras que con la última actualización, la versión de SL4A utilizada, sí es factible.

La conexión en sí se realiza iniciando una actividad de Android en el programa principal de Python. Esta comunicación es bidireccional, regresando por tanto al código Python una vez concluida la tarea escrita en Java.

Se ha diseñado el sistema de forma que una vez realizada una cierta tarea en Python, se transmite toda la información necesaria al lenguaje Java y a partir de ese punto, la aplicación EyeGrade se implementa en Java completamente, de tal manera que solo se realiza una vez la conexión entre lenguajes. Se crea una sola conexión en vez de crear una comunicación cada vez que se necesite emplear la biblioteca OpenCV, mostrar la GUI o tomar una imagen con la cámara.

3.3.2 Traducción de código

La alternativa seleccionada implica la conversión del código escrito para el programa original en Python, a uno diferente como es Java. Esta acción requerida por el diseño final, es viable aunque tediosa, siendo necesario prestar mucha atención a dicha tarea, debido a la gran cantidad de código a traducir y a las diferencias sintácticas que presentan ambos lenguajes.

Aspectos importantes a destacar de este tema son la creación de nuevas estructuras de datos propias o la necesidad de encontrar ciertas bibliotecas específicas para Java sobre Android. Todos estos aspectos serán explicados en el capítulo 5 dedicado a la implementación.

3.3.3 Biblioteca OpenCV

El diseño del modo de uso de la biblioteca OpenCV se ha realizado a través de una biblioteca hecha para lenguaje Java sobre la plataforma Android. En el apartado 5.3.5 se explica mejor cómo se ha resuelto este problema y cómo se ha podido llevar a cabo la alternativa de diseño planteada aquí.

Esta biblioteca es básica para el diseño de esta aplicación, y por consiguiente para su migración entre plataformas. Su gran relevancia en el programa EyeGrade, así como la funcionalidad que aporta al mismo, ha sido uno de los principales motivos de cambio entre la alternativa de diseño expuesta en el apartado 3.2 y la explicada en este propio apartado, basada en el uso conjunto de los lenguajes de programación Python y Java.

3.3.4 Interfaz Gráfica de Usuario

Las herramientas que ofrece Android para llevar a cabo dicha tarea han sido las responsables de que se haya decidido desarrollar el diseño de la GUI siguiendo la estructura habitual de la plataforma Android, aprovechando de esa forma las facilidades que presenta dicha plataforma para realizar una GUI aceptable, tanto los ficheros de *layout* XML como la parte más dinámica a través de Java.

La interfaz gráfica de usuario no ha sido la causante de necesitar el lenguaje de programación Java, pero sí la que ha provocado que se realice todo en código Java a partir de cierto punto del programa.

La afirmación anterior se basa en la no posibilidad de mostrar por pantalla un diseño visual fijo mientras se cambia varias veces entre el lenguaje Python y Java. Esto ha forzado la elección de realizar una sola vez la transmisión entre el código de ambos lenguajes de programación.

3.3.5 Empleo de la cámara

Una vez tomada la decisión de realizar la aplicación (el segmento final del programa en concreto) mediante código Java, se ha de adaptar también la indispensable funcionalidad de la captura de imágenes.

Las únicas funciones que no se han podido utilizar de la biblioteca OpenCV han sido las referentes a la captura de imágenes. Este inconveniente se ha tenido, por tanto, que solventar mediante el uso de las herramientas que facilita Android para el *hardware* de la cámara.

En el apartado 5.3.8 se expone más detalladamente lo realizado con respecto a esta sección, únicamente puntualizar, que de las posibles opciones que se barajaron para realizar esta acción, se optó por la propia creación del código que se encarga de la cámara.

3.4 Conclusiones

Partiendo de un programa inicial escrito en Python y creado para PC, se plantea el diseño realizado para llevar a cabo la migración de dicho programa a su aplicación para Android equivalente.

Al diseño inicial se le presentan una serie de problemas que hacen necesarios algunos cambios. La principal consecuencia surgida de dichos inconvenientes es la necesidad de emplear el lenguaje de programación Java para poder implementar ciertas funcionalidades de la aplicación.

Por tanto, el diseño implementado comparte ambos lenguajes de programación, Python y Java, el del programa original y el usual de Android respectivamente. Con el fin de realizar ciertas funciones específicas de la aplicación, en el diseño final se han tenido que implementar métodos distintos de los originales pero adaptados a la plataforma Android.

Capítulo 4

Diseño del sistema

4.1 Introducción

En este capítulo se va a realizar un análisis del diseño del sistema. Se explicarán con detalle los conceptos básicos de la problemática del sistema a implementar y las tareas precisadas para la migración de la aplicación, presentando posteriormente la arquitectura y los posibles casos de uso del programa.

Como se ha expuesto en el capítulo 3, el objetivo del sistema que se presenta en este trabajo, no es la implementación de una aplicación, sino la migración de una aplicación ya existente para PC, a un dispositivo móvil basado en tecnología Android. La aplicación original, EyeGrade, está escrita en el lenguaje de programación Python, diferente del lenguaje Java usado para implementar las aplicaciones Android. Esta discrepancia entre lenguajes ha sido un punto clave a la hora de adaptar la aplicación. También se han tenido que solventar otros aspectos, como son el hecho de lograr crear un proyecto Android entero que permita escribir la aplicación en Python, con las bibliotecas propias del lenguaje, o el diseño e implementación de la consiguiente interfaz gráfica de usuario.

A la aplicación implementada a lo largo de este trabajo, y diseñada para uso en dispositivos móviles, se le ha dado el nombre de "EyeGrade: visión artificial aplicada al ámbito educativo". La aplicación, basándose en la original (ver apartado 2.6), se caracteriza principalmente por su funcionalidad más importante, la corrección de un examen multirespuesta mediante la cámara del dispositivo móvil Android. Los procesos implícitos en esta acción son, a grandes rasgos, la detección de la imagen de un examen a través de la cámara del dispositivo móvil, el análisis de las respuestas escritas en dicho examen y la presentación por pantalla del examen corregido.

La aplicación se divide según su funcionalidad en dos partes claramente diferenciadas. La creación de exámenes es la primera de ellas, siendo la parte en la que se crean los archivos necesarios tanto para configurar los exámenes como para la configuración del módulo de visión artificial de los mismos. Este proceso es secundario dentro de la aplicación EyeGrade, pero no por ello menos importante. La tarea básica de la aplicación es la que tiene como objetivo detectar con la cámara una imagen de un examen y, mediante su procesado, mostrar en la pantalla del dispositivo móvil dicho examen ya corregido.

Ambas partes de la aplicación reciben una serie de parámetros de entrada por parte del usuario (ver Tabla 2. 10 y Tabla 2. 11). Algunos de estos datos de entrada serán obligatorios y otros optativos, y en función de todos ellos se obtendrán los diferentes ficheros y resultados de salida. Es decir, el usuario realiza la configuración de la aplicación mediante los parámetros de entrada antes de utilizar la aplicación propiamente dicha.

4.2 Tareas necesarias para la migración

Para poder migrar la aplicación original hecha en Python a un dispositivo móvil basado en tecnología Android, se han de cubrir una serie de requisitos necesarios para asegurar el correcto funcionamiento de esta 'nueva' o 'adaptada', aplicación.

A continuación se van a describir las tareas que plantea la migración a Android de la aplicación EyeGrade, enumeradas en la Tabla 4. 1:

Tarea
Creación de una aplicación Android en Python
Adaptación de bibliotecas
Recepción de datos
Creación de la interfaz gráfica de usuario
Adaptación a las limitaciones de la migración
Manipulación de ficheros
Captura de imágenes

Tabla 4. 1: Tareas necesarias para la migración

(1) Creación de una aplicación Android escrita en código Python. Realizar un proyecto completo para la plataforma Android que siga las rutinas escritas en Python, en vez del lenguaje de programación Java, típico de dicha plataforma.

(2) Obtención de distintas bibliotecas empleadas en el código original, para utilizarlas en código Python sobre la plataforma Android. Algunas de estas bibliotecas son: "OpenCV", la biblioteca destinada principalmente para programar funciones de visión artificial, es decir, visión por ordenador en tiempo real. La biblioteca utilizada para la detección de los identificadores de los estudiantes, la biblioteca "TRE", utilizada para la búsqueda de expresiones regulares aproximadas, o por ejemplo, la biblioteca "CSV" (*Comma-Separated Values*), empleada en el tratamiento de los ficheros con extensión `.csv`.

(3) Creación de una interfaz de usuario nueva con la que recibir los datos de entrada por parte del usuario, que originalmente se introducían a través de la línea de comandos del ordenador.

(4) Creación de una nueva interfaz gráfica de usuario con la que mostrar los resultados e interactuar con el usuario de la forma más parecida posible a la original dentro del nuevo entorno móvil.

(5) Consideración de las limitaciones (según su resolución) que existen a la hora de captar o mostrar imágenes en las pantallas de los dispositivos Android.

(6) Gestión de carpetas y ficheros, presentes en el dispositivo móvil, mediante código Python. Adaptación de código para el acceso y manipulación de los archivos que originalmente se encontraban en un ordenador y ahora están en el terminal Android.

(7) Toma de imágenes a través de la cámara del dispositivo móvil Android. Adaptación de esta funcionalidad del programa

4.3 Arquitectura de la aplicación

La arquitectura del sistema implementado, es una arquitectura sencilla y básica. Si se piensa sólo en la creación de exámenes, y no en la visualización, el sistema no requeriría más que el dispositivo Android. Sin embargo, la principal función y utilidad básica de la aplicación, es la de visualizar y corregir exámenes de manera fácil y sencilla a través de la cámara. Por tanto, será este caso (visualización y análisis de la imagen), el que se tome como sistema general de la aplicación EyeGrade.

En la Figura 4. 1 se presenta la arquitectura del sistema general, la cual está formada por el dispositivo móvil Android y la hoja de examen a analizar.

Figura 4. 1: Arquitectura del sistema

Como se puede comprobar, esta arquitectura no conlleva un alto nivel de complejidad. Los únicos puntos destacables de la estructura presentada en la Figura 4. 1, son los prerrequisitos necesarios de los que debe disponer el terminal móvil Android para poder realizar con éxito sus funciones.

El condicionamiento del dispositivo Android se basa en los siguientes grandes bloques:

- **Cámara de fotos**: dispositivo móvil Android que disponga de una cámara integrada (ver apartado 5.3.8). Preferiblemente debe estar situada en la parte opuesta a la pantalla, aunque esto no es estrictamente necesario.

- **Versión mínima 2.2** de Android (nivel 8 de API): por motivos relacionados con la migración entre lenguajes, no es posible la utilización de versiones anteriores[9].

- **Aplicaciones extras** instaladas: además de la aplicación en sí (EyeGrade), se requiere de una serie de aplicaciones extras, que deben ser instaladas antes de poder ejecutar el sistema. Una de ellas permitirá trabajar dentro del terminal con ficheros externos, mientras que otras permitirán ejecutar la aplicación (ver apartado 5.3.4).

La comunicación entre los elementos presentes en la arquitectura de la Figura 4. 1 es, al igual que el conjunto del sistema, sencilla. En el caso de la comunicación existente entre usuario y terminal móvil, ésta se realiza a través de la pantalla táctil del dispositivo. Por otra parte, la conexión entre terminal y examen se realiza gracias a la ya mencionada cámara digital del dispositivo. Para poder realizar cualquier actividad este sistema necesita la existencia, dentro del terminal, de un conjunto de ficheros (ver apartado 5.3.2).

[9] La versión de Android viene impuesta por el uso de la biblioteca JavaCV, utilizada para implementar los métodos de OpenCV originales de Python (ver apartado 5.3.5).

4.4 Diagramas de casos de uso

En este apartado se van a explicar las diversas acciones que puede realizar el usuario con el sistema. Estas funcionalidades, también llamadas casos de uso, se representan en el diagrama de la Figura 4. 2. En dicha representación se muestra gráficamente los usos que pueden darse de la aplicación por parte del usuario.

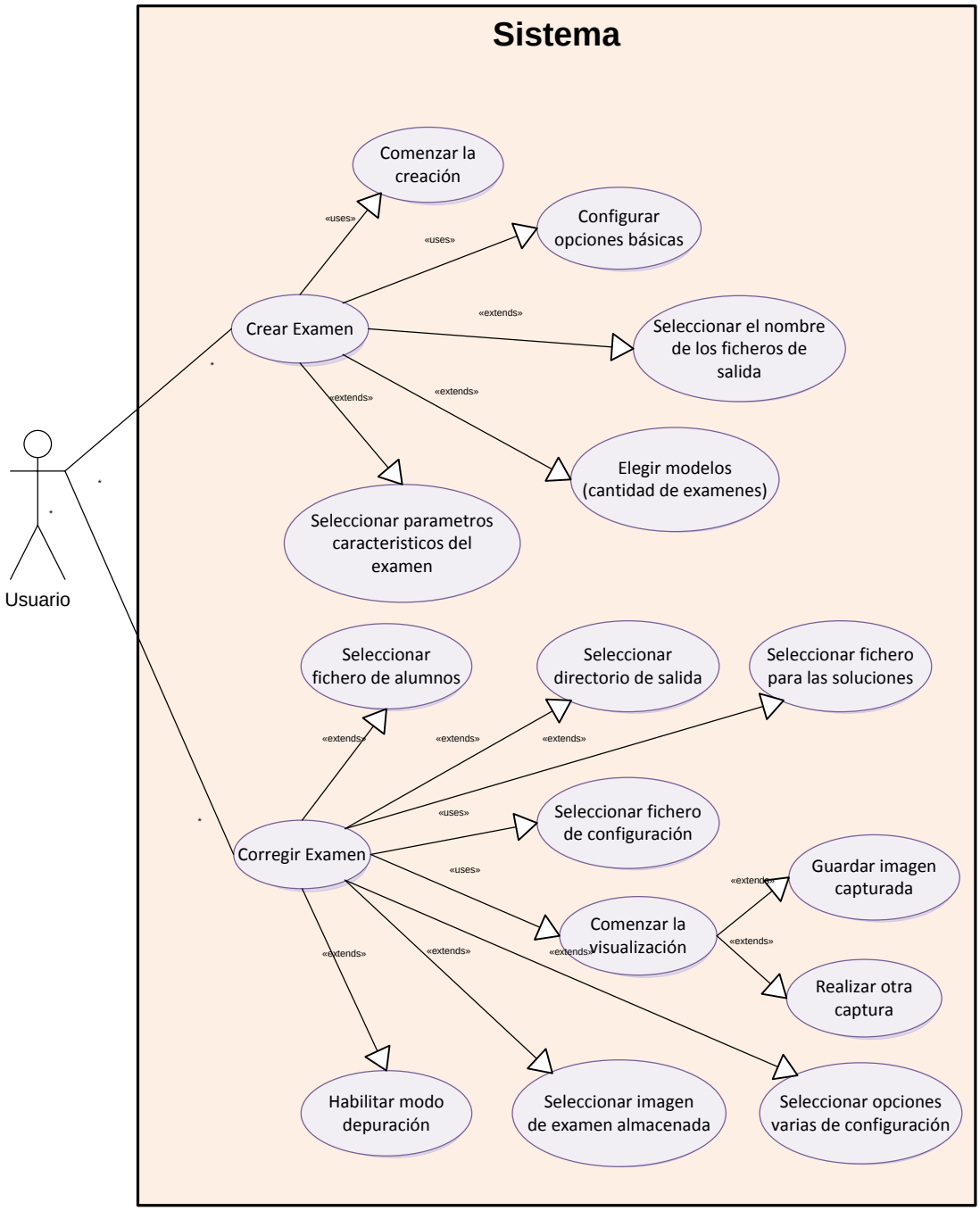


Figura 4. 2: Diagrama de casos de uso

Como se puede observar en la Figura 4. 2, la aplicación se basa en dos casos de uso principales, o utilidades básicas, y dentro de ellos surgen otras posibles acciones dependientes de los mismos.

La creación de los ficheros de un examen, y la visualización y análisis de un examen en formato papel a través de la cámara, son los dos casos primordiales de EyeGrade. Estas dos opciones se representan en el diagrama mediante las flechas que salen del usuario y que significan que el usuario al iniciar la aplicación accede al menú de creación de exámenes, o al de corrección de los mismos.

Caso de uso básico 1: creación de exámenes.

La acción que realiza en este caso el usuario tiene como finalidad la creación de un fichero de texto plano (con formato Latex) que contendrá los datos necesarios para la creación de una hoja de examen, es decir, de un archivo PDF que será el examen en sí. Este caso de uso no tiene únicamente como objetivo la creación del fichero Latex del enunciado, sino que posee otro más, la creación del fichero de configuración (con formato `.eye`) necesario para realizar la corrección de dicho examen.

Los casos de uso relativos que se les presentan a los usuarios cuando seleccionan la opción de crear un examen se detallan en la Tabla 4. 2. Estas distintas posibilidades de acción se le presentan al usuario en forma de opciones de un menú gráfico.

Nombre	Descripción
Comenzar la creación	Iniciar la creación de los exámenes y posterior almacenamiento de los ficheros en el dispositivo móvil Android.
Configurar opción básicas	Elección de parámetros obligatorios para la realización del examen, teniendo en cuenta que todos los ficheros deben residir en el interior del dispositivo al iniciarse la aplicación. - Selección del fichero con formato Latex que será la plantilla del examen. - Selección del fichero XML contenedor de los datos del examen, o bien elección de los dos parámetros indispensables para el examen, numero de preguntas y número de respuestas posibles para una pregunta.
Seleccionar el nombre de los ficheros de salida	Elección del nombre identificativo a utilizar en la creación de los ficheros de salida Latex y `.eye`. Esta selección implica la decisión, a su vez, de realizar la creación o no, de los ficheros de configuración `.eye`.
Elegir modelos (cantidad de exámenes)	Introducción de las letras identificativas que indican la cantidad de exámenes que se crearán. Cada uno de dicho exámenes se corresponderá con cada una de las letras introducidas.
Seleccionar parámetros característicos del examen	Posible introducción de múltiples opciones propias del examen, entre las que se encuentran fecha, nombre de la asignatura o duración del tiempo del examen.

Tabla 4. 2: Casos de uso derivados de la creación de exámenes

Caso de uso básico 2: corrección de exámenes.

El objetivo que se pretende facilitar con esta función de la aplicación es el de visualizar, en la pantalla del dispositivo móvil, la imagen del examen, junto con las anotaciones del análisis de éste. Es decir, se mostrarán cuales son las soluciones correctas, las incorrectas y la puntuación total.

Otra información opcional que ofrece esta funcionalidad de la aplicación es la referente al alumno que ha realizado el examen. Esta opción dependerá del fichero de configuración de entrada y del fichero que contiene la lista de alumnos.

Otros objetivos que presenta el actual caso de uso es prestar al usuario las opciones de mostrar la imagen en el visor con cierta información útil para su depuración, y la posibilidad de guardar en el dispositivo móvil la imagen del examen con todo el análisis ya hecho.

Al igual que se ha hecho en el primer caso de uso básico, en la Tabla 4. 3 se describen las opciones que se derivan de esta funcionalidad de corrección,.

Nombre	Descripción
Comenzar la visualización	Visionado de la imagen del examen ya procesado, habilitando las diversas opciones como son, por ejemplo, "guardar" las capturas en el dispositivo, o "descartar" la imagen tomada y realizar una nueva captura.
Seleccionar fichero de configuración	Elección del fichero de configuración propio de la aplicación (formato `.eye`), existente en el dispositivo y correspondiente con la hoja de examen que se pretende analizar.
Seleccionar fichero de alumnos	Elección del fichero que contiene la lista de los posibles alumnos, con los números identificativos y sus correspondientes nombres.
Seleccionar directorio de salida	Elección de la carpeta del dispositivo móvil Android donde almacenar los ficheros obtenidos.
Seleccionar fichero para la soluciones	Designación del archivo específico donde almacenar la información analizada.
Habilitar modo depuración	Habilitación o no de las herramientas que permiten depurar las imágenes capturadas con la cámara, modo *Debug*.
Seleccionar imagen de examen almacenada	Análisis de una imagen ya almacenada en el terminal, en lugar de analizar una captura de examen realizada con la cámara.
Seleccionar opciones varias de configuración	Introducción opcional de una serie de parámetros como son el umbral numérico del procesado de la imagen, o la posibilidad de habilitar la captura manual de imágenes.

Tabla 4. 3: Casos de uso derivados de la corrección de exámenes

4.5 Funciones del programa

El funcionamiento interno que seguirá el programa para lograr realizar los objetivos pretendidos por el usuario se puede separar en dos grandes bloques: creación y corrección.

El primer conjunto de código seguirá, a grandes rasgos, los siguientes pasos de funcionamiento:

(1) Comprobación de los parámetros de configuración introducidos por el usuario y verificación de la no existencia de incongruencia en los datos de entrada.

(2) Configuración, preparación y creación del objeto encargado de la maquetación del examen.

(3) Creación de los múltiples ficheros Latex en función de los modelos de examen indicados por el usuario.

(4) Creación y almacenamiento del fichero de configuración (con formato .eye), necesario para la visualización y análisis por parte de EyeGrade. Este último paso depende de lo introducido por el usuario.

Con respecto a la visión artificial de la imagen del examen, los pasos esquemáticos, al igual que se ha hecho anteriormente, serían los descritos a continuación:

(1) Comprobación de los parámetros de configuración introducidos por el usuario y verificación de la no existencia de incongruencia en los datos de entrada.

(2) Configuración y preparación de los ficheros y opciones de procesado de las imágenes a tratar.

(3) Inicialización de la interfaz gráfica de usuario.

(4) Captura de una imagen a través de la cámara u obtención de la imagen almacenada según los designios del usuario.

(5) Procesado de la imagen. Detección de las tablas con las preguntas del examen y respuestas escritas por el alumno.

(6) Presentación de la imagen procesada en caso satisfactorio, o búsqueda de una nueva captura de examen en caso contrario.

(7) Modo en espera a la acción del usuario y procediendo posteriormente en función de las instrucciones de éste.

4.6 Diagramas de flujo

El sistema implementado consta de dos funcionalidades claramente diferenciadas y comentadas a lo largo de todo este trabajo: creación de ficheros de examen, y corrección de exámenes a través de visión artificial.

El acceso a cada una de ellas se realiza mediante un menú inicial, donde el usuario decide que tarea desea realizar (ver Figura 4. 3).

Este menú principal es simple y sencillo, representando la raíz de la aplicación y compuesto por los botones correspondientes a cada una de las partes.

El menú principal representado en la Figura 4. 3, es un nuevo elemento no existente en el programa creado para PC, pero necesario en la plataforma Android. Por tanto, este menú inicial no es una simple adaptación de la aplicación original. Este nuevo componente de la aplicación tiene su razón de ser en la necesidad de diferenciar la tarea que el usuario quiere realizar

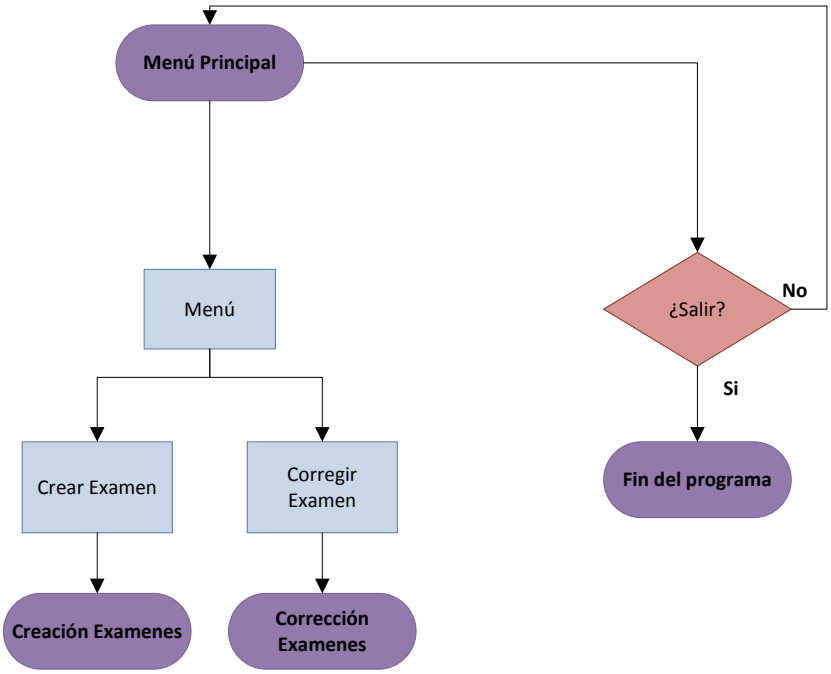


Figura 4. 3: Diagrama de flujo del menú principal

El menú inicial representado por el diagrama de flujo de la Figura 4. 3 será el nexo de unión de las dos funciones básicas implementadas en la aplicación EyeGrade. Este menú será accesible por los menús de configuración de ambas partes de la aplicación.

Una vez pulsada la opción elegida, y antes de comenzar con los diagramas presentes en la Figura 4. 4 o la Figura 4. 6, según haya sido la selección, aparecerá una pantalla que informa al usuario de la aplicación de los parámetros mínimos que necesita dicha funcionalidad.

4.6.1 Creación de exámenes

Tarea encargada de la creación de los ficheros de examen Latex y de los ficheros `.eye` de configuración. Este proceso se divide en dos diagramas de flujo, presentados en la Figura 4. 4 y en la Figura 4. 5. Esta separación es debida a que la tarea de creación se subdivide a su vez, en configuración de parámetros para la creación de ficheros (primer diagrama) y en la creación en sí misma (segundo diagrama).

Una vez que se ha pulsado la opción de 'Creación' del menú de la Figura 4. 3, aparece una ventana donde se presenta el menú de la creación de exámenes que sigue el funcionamiento planteado en el diagrama de la Figura 4. 4.

En él se presenta una lista de opciones que dejan al usuario elegir el parámetro a introducir. Pulsando sobre cualquiera de ellas, aparecerá una ventana informativa del dato de entrada elegido. Posteriormente, y dependiendo de la opción elegida, aparecerá una pantalla donde se tendrá:

- Un campo de texto donde introducir algún número o cadena de texto (como por ejemplo el número de preguntas, la fecha y los modelos del examen).
- Una pantalla donde se navegará a través del dispositivo para seleccionar algún directorio o fichero (por ejemplo el fichero de configuración).
- Dos botones que dejan activar o no la opción *shuffle*.

En el menú, junto a las opciones de los distintos parámetros, se tiene el botón que dará comienzo a la creación propiamente dicha. Una vez pulsado dicho botón y ratificada esta decisión, el programa realiza la comprobación de que al menos se han introducido los datos mínimos.

Los datos mínimos necesarios para poder realizar los ficheros de examen son los siguientes:

- La plantilla del examen (fichero *template*).
- La información mínima del examen. El usuario puede introducir esta información mínima de dos distintas formas alternativas a la vez que mutuamente excluyentes:

  (a) Indicando el número de preguntas y el número soluciones del examen.
  (b) A través de un fichero de configuración que contiene en su interior la información del examen.

Una vez confirmado el deseo de comenzar con la creación de los exámenes, se realiza la comprobación de la introducción de los requisitos mínimos. Tal verificación se presenta en el diagrama de flujo de la Figura 4. 4.

El final del diagrama de flujo presente en la Figura 4. 4, 'Creación de los ficheros de exámenes', es el punto de comienzo del diagrama de flujo representado en la Figura 4. 5. Cuando dé comienzo este último, significará que el usuario ha introducido al menos los parámetros mínimos y finalizará aquí la relación inicial del usuario con el programa.

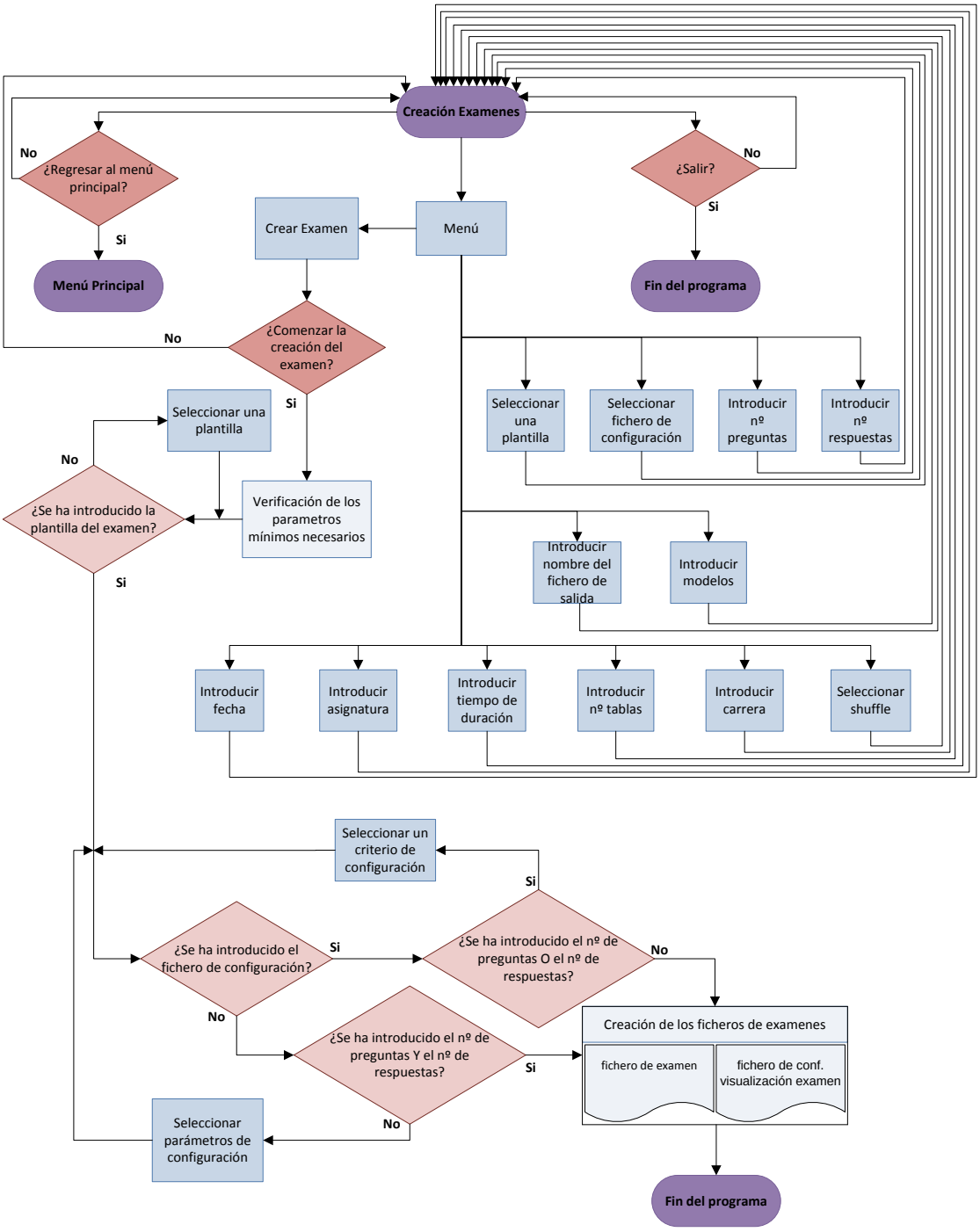


Figura 4. 4: Diagrama de flujo del menú de creación de exámenes

Una vez que la verificación de la información es positiva, se procede a realizar la creación de los ficheros.

Nótese que en el proceso de entrada de datos, no se realiza comprobación alguna de que los datos sean correctos, tal verificación se realiza posteriormente a lo largo del proceso mostrado en la Figura 4. 5, y según vayan siendo utilizados.

La única comprobación realizada, además de la selección de los parámetros mínimos necesarios, tiene lugar a la hora de verificar que los datos, que forzosamente tienen que ser numéricos, cumplan tal restricción. En caso de no ser así, se le informa al usuario de la necesidad de introducir un valor numérico y se le vuelve a solicitar el dato.

La tarea de creación de ficheros de examen, una vez recibidos los datos por parte del usuario (ver diagrama de flujo de la Figura 4. 5), tiene como objetivo final la realización de los archivos Latex correspondientes con las hojas de exámenes y a los ficheros `.eye` dirigidos a la configuración de la segunda parte de esta aplicación, que tiene como objetivo el análisis visual del examen.

Una vez comenzado el proceso de creación se tendrán en cuenta los datos introducidos por el usuario siguiendo los siguientes pasos:

(a) Tomando la información mínima del examen (datos del fichero de configuración ó valores introducidos por el usuario).

(b) En caso de haberse metido alguna característica más del examen, esta será también guardada por el programa antes de comenzar con los ficheros de salida.

(c) Asignación de los nombres a los ficheros de salida.

(d) En función del número de modelos que se hayan introducido, se irán creando los ficheros de examen con formato Latex[10].

(e) Si se ha introducido un valor para la salida de ficheros, se creará el fichero `.eye`, necesario para la corrección de exámenes.

Las diferencias existentes en los tonos de los diagramas presentados, tienen relación con la interacción directa o indirecta con los usuarios:

- Tonos fuertes: el usuario es el que decide o realiza una acción.
- Tonos débiles: decisiones o acciones correspondientes a la aplicación.

Esta idea se ha desarrollado a lo largo de todos los diagramas de flujos presentes en este apartado. Un ejemplo concreto con el que se puede apreciar esto, tiene lugar en el diagrama presente en la Figura 4. 4, donde en la pregunta: “¿Comenzar la creación del examen?”, que se encuentra en tono fuerte, el que contesta a esta pregunta es el usuario. Según lo que marque el usuario en la pantalla, así se procederá.

El código que lleva a cabo todo este proceso está escrito en código Python. La tarea descrita en el diagrama de la Figura 4. 4, es código creado explícitamente para esta adaptación a la plataforma Android. Es una nueva interfaz gráfica de usuario creada para hacer operativa la aplicación EyeGrade en dispositivos móviles Android. En el apartado 5.3.3 se explica más detalladamente aspectos concretos de dicha GUI.

El código que describe el diagrama de flujo de la Figura 4. 5 y que está escrito en Python, es prácticamente el mismo que el del programa original para PC. Adaptado en ciertos aspectos para ser compatible con la nueva plataforma, pero manteniendo gran parte del núcleo básico del código.

[10] El mínimo número de ficheros será uno, al menos, el modelo ‘A’ (valor por defecto) será creado.

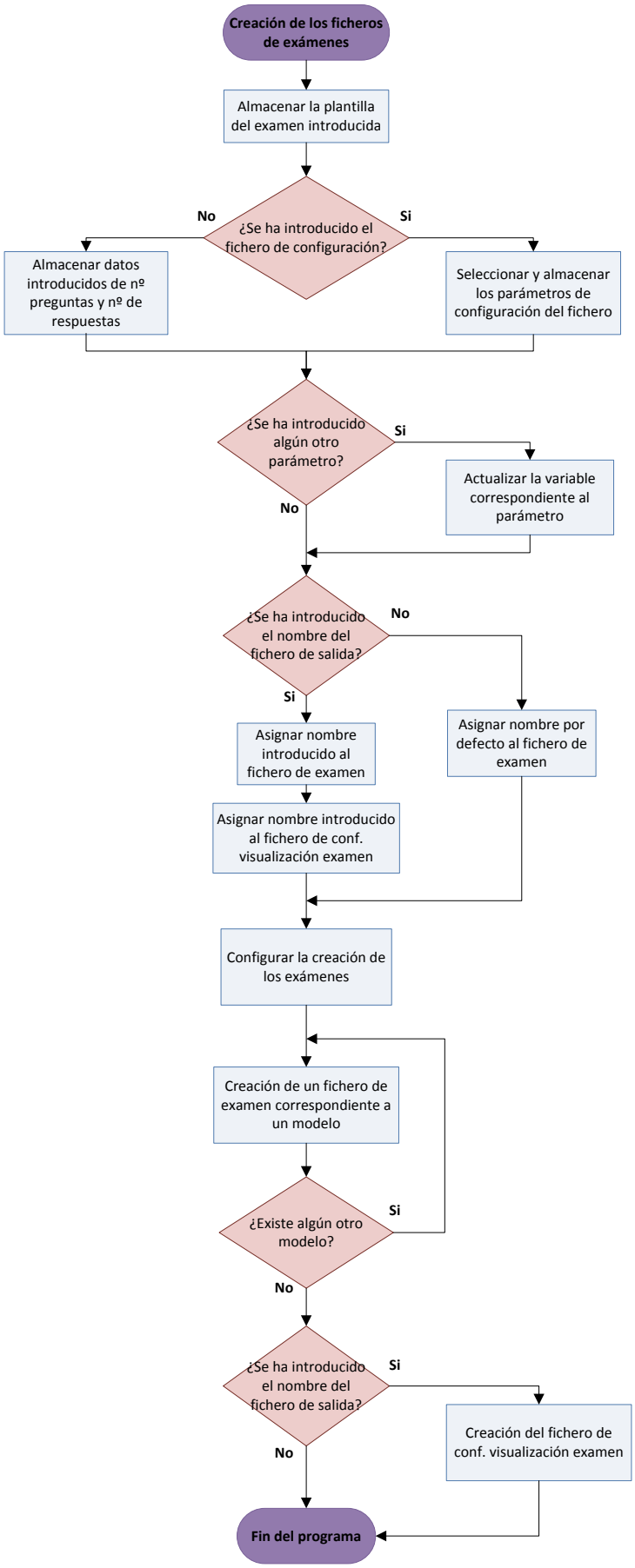


Figura 4. 5: Diagrama de flujo de la creación de exámenes

4.6.2 Corrección de exámenes

Tarea encargada del análisis y procesado de las imágenes de exámenes, junto con su posterior presentación gráfica de resultados. Este proceso se divide en dos diagramas de flujo presentados en la Figura 4. 6 y en la Figura 4. 7.

En el primero de ellos, el diagrama de la Figura 4. 6, tiene lugar la recepción de parámetros de configuración por parte del usuario, siendo unos datos de entrada obligatorios y otros opcionales. Mientras, en el segundo diagrama (el presente en la Figura 4. 7) se presenta el proceso que realiza el programa para: capturar la imagen, analizarla, obtener resultados y mostrarla posteriormente al usuario.

Una vez que se ha pulsado la opción de 'Corrección' del menú de la Figura 4. 3, aparece una ventana donde se presenta el menú de configuración de la corrección de exámenes. Dicho menú de configuración sigue el funcionamiento planteado en el diagrama de la Figura 4. 6.

En el menú previo a la visualización de los exámenes, se presenta una lista de opciones que dejan al usuario elegir el parámetro de configuración a introducir. Pulsando sobre cualquiera de las opciones aparecerá, antes de dar paso a la pantalla correspondiente de cada parámetro, una ventana informativa del dato de entrada elegido.

Dependiendo de la opción elegida en el menú representado en la Figura 4. 6, se puede obtener:

- Un campo de texto donde introducir algún número (como por ejemplo el identificador).
- Una pantalla donde se navegará a través del dispositivo para seleccionar algún directorio o fichero (por ejemplo el directorio de salida).
- Dos botones para activar o no alguna opción de configuración (ajuste manual o modo depuración por ejemplo).
- Un menú desplegable sobre el que seleccionar alguna variable (ejemplo del valor para el umbral de detección).

Además de éstas, hay una variable (elección de imagen) en la que si se selecciona, se debe elegir entre dos tipos de imágenes diferentes.

En el menú, junto a las opciones de los distintos parámetros, se tiene el botón que dará comienzo a la visualización artificial del examen. Una vez pulsado dicho botón y ratificada esta decisión, el programa realiza la comprobación de que, al menos se ha introducido el parámetro mínimo de configuración, el fichero (con formato `.eye`) propio del examen que se quiere corregir.

Una vez que la verificación de la información es positiva (se ha introducido un fichero de configuración), se procede a realizar la visión artificial del examen. Es decir, se concluye con la actividad descrita en el diagrama de flujo de la Figura 4. 6 y se comienza con el presente en la Figura 4. 7, procediendo por tanto al comienzo de la corrección del examen.

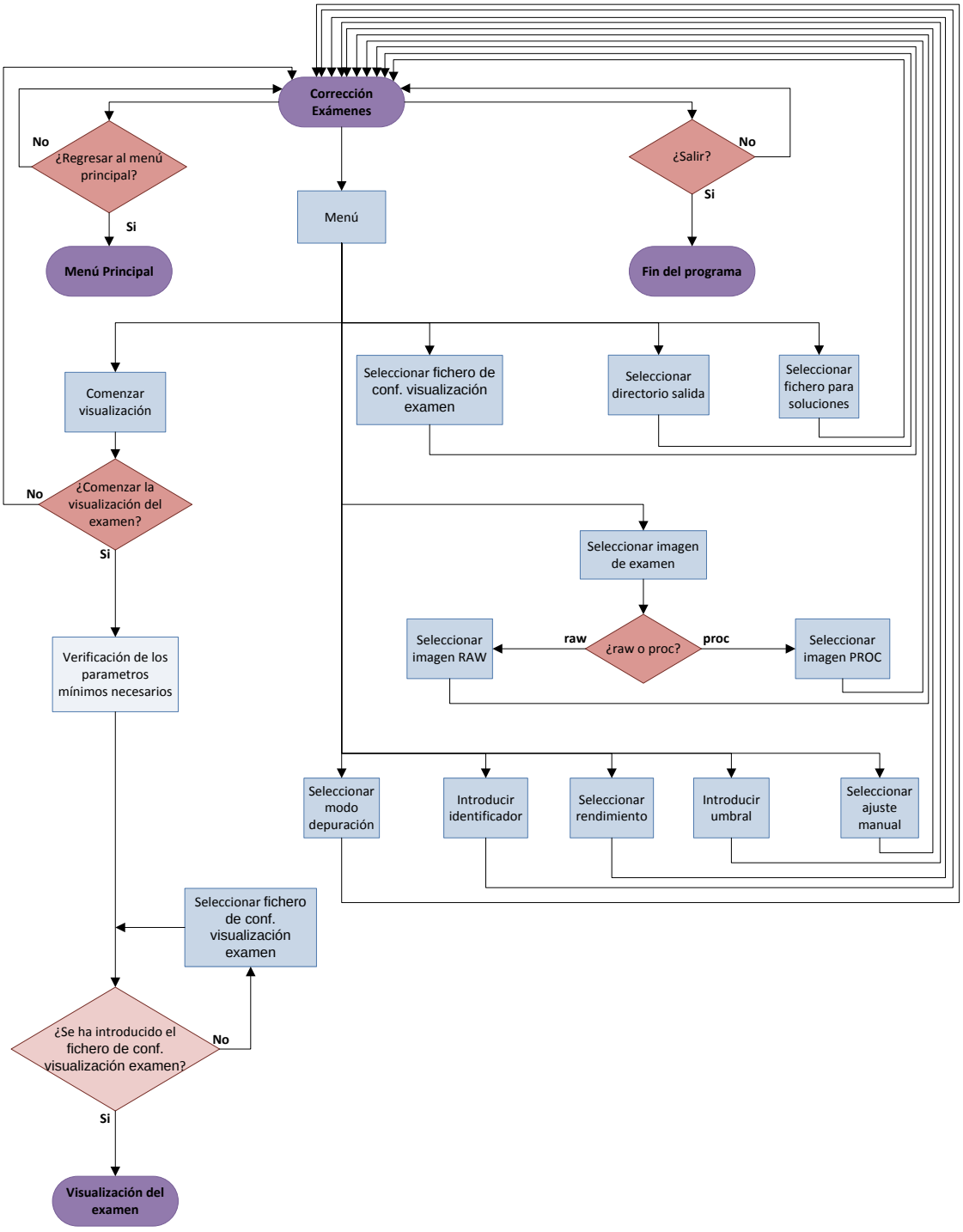


Figura 4. 6: Diagrama de flujo del menú de corrección de exámenes

La tarea de corrección de un examen una vez recibidos los datos por parte del usuario (ver diagrama de flujo de la Figura 4. 7), tiene como objetivo la obtención de una imagen, su procesado en función de la configuración y la representación de dicha imagen junto con el análisis realizado.

Hasta llegar a este punto de la aplicación, se había implementado todo el programa con lenguaje Python. Sin embargo, a partir de este momento (salvo la sección de verificación de parámetros, como se indica en la Figura 4. 7) la implementación se realizará en lenguaje Java y haciendo uso del fichero *layout* XML característico de esta aplicación (el archivo `eyegrade.xml`).

Una vez comenzado el proceso de corrección en función de los datos de configuración del examen a procesar (que son los que indican cómo realizar la corrección del mismo) se realizará la conexión con el código Java:

(a) Almacenamiento de los datos de entrada.

(b) Iniciación de la GUI propia de la aplicación.

(c) Captura de una imagen de examen.

(d) Procesamiento de la imagen.

(e) Mostrar indefinidamente el análisis realizado a la espera de una acción por parte del usuario.

A través de los botones del menú derecho de la pantalla, el usuario podrá realizar la acción que desee entre las siguientes:

-1- Guardar la imagen del examen corregida así como el fichero con las soluciones.

-2- Descartar la captura y realizar una nueva.

-3- Depurar la imagen, si el modo depuración está activado.

-4- Modificar algún detalle de la captura que sea erróneo.

-5- Salir de la aplicación.

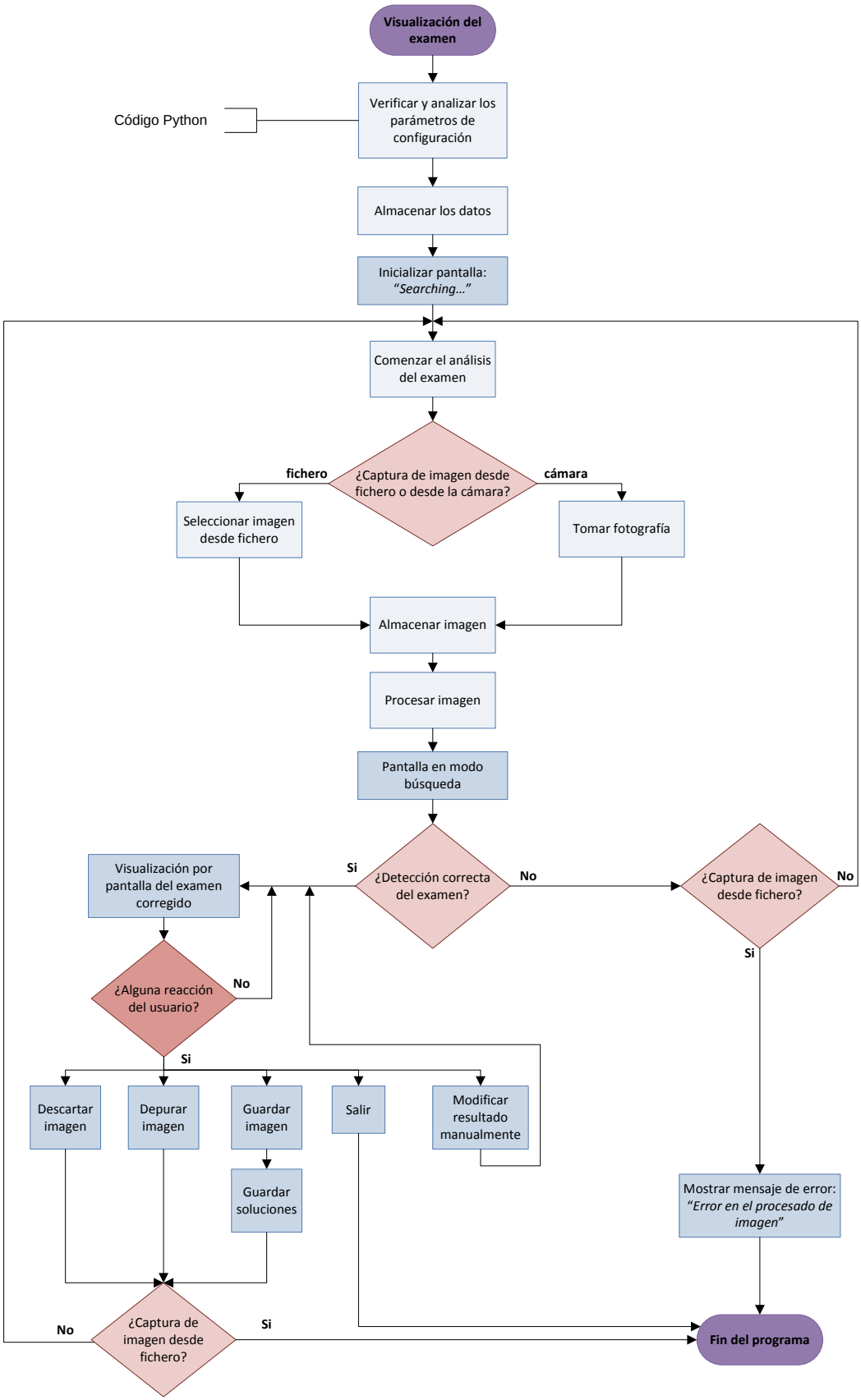


Figura 4. 7: Diagrama de flujo de la visualización de exámenes

4.7 Diagramas de clases

Este apartado, dedicado a los diagramas de clases, está exclusivamente centrado en la funcionalidad principal de la aplicación EyeGrade. Es decir, la parte dedicada a la visualización artificial de los exámenes, la parte implementada en Java.

En la Figura 4. 8, se presenta un diagrama esquemático de las clases que intervienen en el proceso. Unas de ellas serán parte básica de la ejecución (ver Figura 4. 9), mientras que otras forman parte del conjunto de estructuras de datos creados explícitamente para la aplicación (ver Figura 4. 10).

A partir de la estructura presentada en la Figura 4. 8 se puede observar cómo la *Activity* principal de EyeGrade es `MainActivity`. Aun siendo la primordial, no es la única, puesto que se tienen también `DialogEyeGradeActivity` y `DigitEyeGradeActivity`.

Esta última actividad no se usa en la versión actual de la aplicación. `DigitEyeGradeActivity` se usa cuando está activa la opción de analizar el identificador de los estudiantes. Con ésta actividad se puede introducir el valor numérico que sea preciso para la correcta detección del alumno que ha realizado el examen.

Al no haber sido posible terminar dicha funcionalidad de la aplicación, dicha actividad no será empleada. Pero al estar todo el código dispuesto para ejecutarse, una vez se consiga la biblioteca solicitada (ver apartado 7.3), esta clase, así como el resto del código que hace posible la detección de los alumnos, está implementado en la aplicación actual EyeGrade.

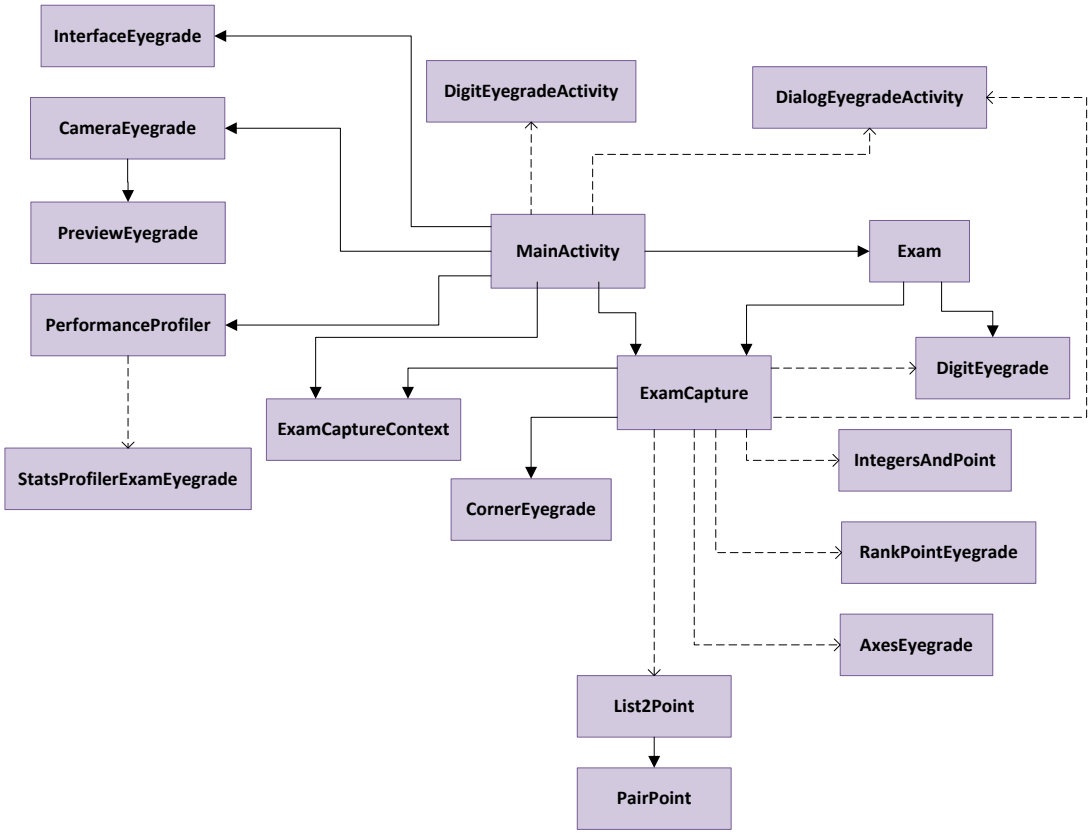


Figura 4. 8: Estructura general del diagrama de clases

De la Figura 4. 8, las clases que componen la funcionalidad principal del programa son las siguientes:

(a) La clase `MainActivity` que extiende de la interfaz `Activity`.

(b) La clase vinculada a la interfaz gráfica de usuario, `InterfaceEyegrade`.

(c) Las clases relativas a la captura de imágenes a través de la cámara, `CameraEyegrade` y `PreviewEyegrade`.

(d) Las clases relacionadas con la imagen del examen y su análisis, `Exam`, `ExamCapture`, `ExamCaptureContext` y `PerformanceProfiler`.

Todas estas clases se pueden observar con más detalle en el diagrama representado en la Figura 4. 9. Intentando no dejar el diagrama demasiado recargado, se ha dejado éste a su vez, de nuevo esquematizado. El número de atributos y métodos de cada una de las clases que aparecen en el diagrama de clases, es sumamente elevado. Este hecho haría que el diagrama fuese difícilmente representable en una hoja, a la vez que lioso e improductivo.

Por lo tanto, los atributos y métodos que se han indicado en el diagrama, son los elementos más útiles a la hora de ver las relaciones de unas clases con otras. Es decir, los atributos que muestran los objetos de otras clases que se poseen, o los métodos públicos llamados desde una clase ajena.

Ejemplos de la simplificación de los componentes se encuentran en: los atributos de `MainActivity` o en los métodos de la clase `Exam`.

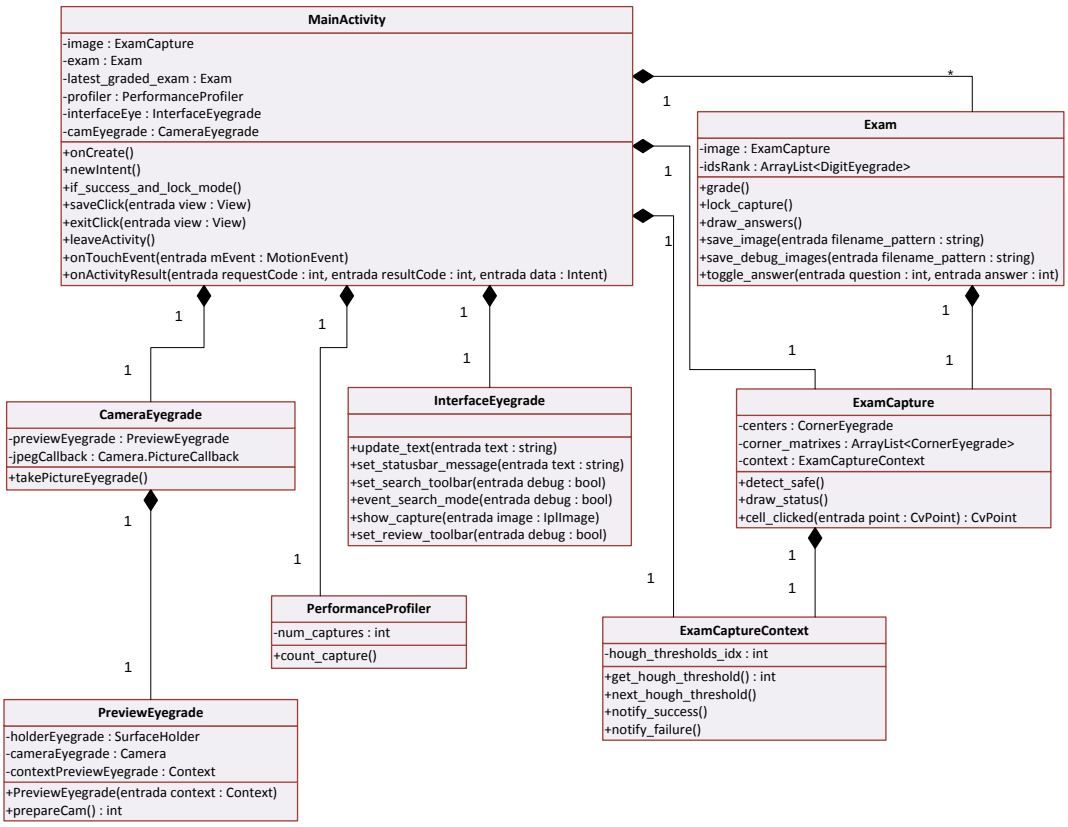


Figura 4. 9: Diagrama de clases núcleo básico

Las ocho clases presentes en el diagrama de clases de la Figura 4. 9 son explicadas a continuación:

(1) **MainActivity**

`MainActivity` es una clase que posee una gran cantidad de atributos relacionados con la configuración de la aplicación (además de los mostrados en la Figura 4. 9), así como los métodos relacionados con el flujo del programa (por ejemplo `if_success_and_lock_mode`) y con el control de los botones que se muestran al usuario por la pantalla (ejemplo `exitClick`). La clase `MainActivity` es el nexo de unión de todas las clases presentes en la Figura 4. 9, es el eje conductor de la aplicación EyeGrade.

(2) **InterfaceEyeGrade**

`InterfaceEyeGrade`, a pesar de no mostrar atributos en el diagrama de la Figura 4. 9, posee un atributo por cada elemento visual de la interfaz gráfica de usuario. Tanto los botones, los campos de texto o la imagen del examen serán controlados por ella. Esta clase es la encargada de gestionar la GUI de la aplicación.

(3) **CameraEyegrade**

A través de la clase `CameraEyegrade` se realizará la captura de imágenes. Es una clase no existente en el programa original al llevarse a cabo la toma de imágenes a través de la biblioteca OpenCV. El método `takePictureEyegrade` será el encargado de realizar dicha función en la aplicación de Android.

(4) **PreviewEyegrade**

La clase `PreviewEyegrade` es un atributo indispensable de la clase `CameraEyegrade`. Sus atributos manejan el control del *hardware* de la cámara de Android y sus métodos gestionan la inicialización de la cámara.

(5) **ExamCapture**

La clase `ExamCapture` está dedicada especialmente al procesado de las imágenes y posee un alto número de métodos internos. Entre todos estos métodos se hallan las funciones relacionadas con la identificación de los estudiantes, la detección de las tablas de los exámenes y su posterior detección de respuestas, o las funciones vinculadas con el hecho de dibujar los análisis realizados sobre la captura.

(6) **ExamCaptureContext**

La clase `ExamCaptureContext` posee información relacionada con el umbral usado en el procesamiento de las imágenes.

(7) **Exam**

La clase `Exam`, proviene del programa original y representa los exámenes que se van a analizar por la aplicación. Contiene un gran número de métodos internos que hacen posible realizar funciones como: calificar un examen (`grade`), dibujar las soluciones encontradas (`draw_answers`) o guardar el examen en el dispositivo Android (`save_image`).

(8) **PerformanceProfiler**

Clase vinculada con el rendimiento obtenido en la captura de un examen con éxito.

El diagrama de clases presentado en la Figura 4. 10 simboliza las clases auxiliares creadas para la aplicación, y que representan las estructuras de datos necesarias para convertir el código Python original al nuevo escrito en Java.

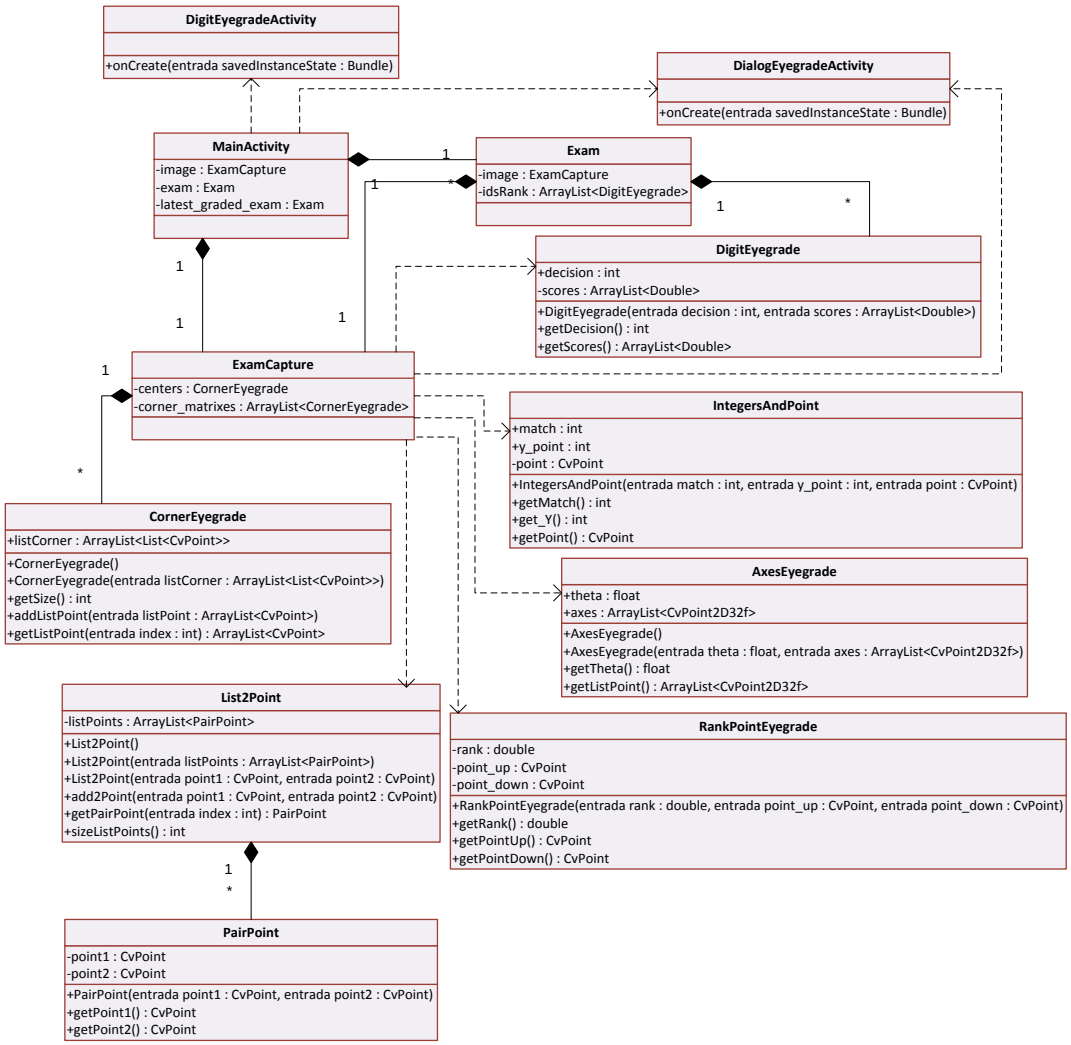


Figura 4. 10: Diagrama de clases estructuras de datos extras

Para poder representar este esquema ha sido necesaria la presencia de tres de las clases ya explicadas, `MainActivity`, `Exam` y `ExamCapture`, pero que son necesarias al ser las que hacen uso de dichas estructuras de datos nuevas y propias de EyeGrade.

En este diagrama también figuran las dos clases que extienden de la interfaz `Activity`. Aun no siendo estructuras de datos nuevas para la aplicación, se ha querido presentar en este esquema, debido a que son clases empleadas en dar soporte al núcleo principal, al igual que el resto de las clases de los nuevos tipos de datos.

El resto de estructuras, presentes en el diagrama de clases de la Figura 4. 10, muestran todos sus atributos y métodos. Elementos todos ellos sencillos, pero indudablemente necesarios para lograr pasar el código Python al nuevo programa implementado en Java para la plataforma Android.

4.8 Conclusiones

Con el fin de conseguir una aplicación autónoma, no dependiente de factores externos, se ha diseñado una arquitectura del sistema sencilla formada únicamente por el terminal Android y la hoja de examen a calificar.

La aplicación EyeGrade posee dos casos de uso principales: creación y corrección de exámenes. Dentro de cada uno de ellos existirán distintas funcionalidades relativas, como pueden ser la elección de los modelos de un examen en la creación de éstos, o la decisión de realizar la calificación de un examen a través de la cámara o mediante una imagen ya almacenada.

Cada una de estas dos funcionalidades diferentes del programa consta de dos diagramas de flujo, uno representa el proceso de obtención de información inicial, y el segundo el proceso que realiza la aplicación para llevar a cabo la tarea concreta.

En el diseño del programa intervienen diversas clases, algunas de ellas representan el conjunto de estructuras de datos auxiliares y otras son las encargadas de ejecutar con éxito la función principal del programa. La clase más importante dentro de este último conjunto es `MainActivity`, actividad conductora de toda la aplicación.

Capítulo 5

Implementación

5.1 Entorno de desarrollo

Eclipse es el entorno de desarrollo utilizado en este proyecto para llevar a cabo la implementación de la aplicación Android. Eclipse es una comunidad de código abierto cuyos proyectos están enfocados en crear una plataforma de desarrollo de software libre. Plataforma compuesta por herramientas, marcos de trabajo y entornos de ejecución para, la construcción, el despliegue y la gestión del *software* a lo largo de su ciclo de vida. El Proyecto Eclipse fue creado originalmente por IBM (*International Business Machines*) en Noviembre del 2001. Posteriormente fue albergado por la Fundación Eclipse, creada en Enero del 2004, dicha fundación es una corporación independiente sin ánimo de lucro que, aparte de encargarse del Proyecto Eclipse, tiene entre sus objetivos fomentar, tanto una comunidad de código abierto, como un ecosistema de productos y servicios complementarios [28].

Según el punto de vista con el que se mire, Eclipse puede definirse de distintas maneras. Eclipse puede definirse como un entorno de desarrollo libre para Java. Es un entorno flexible para experimentar con los nuevos lenguajes de programación, o ampliaciones de los ya existentes. También se puede ver como un marco de trabajo global, en el que se integran muchos modernos y avanzados diseños *software* y técnicas de implementación.

Eclipse se caracteriza por ser un entorno de desarrollo integrado, abierto y una plataforma:

- **Abierto**: Eclipse es abierto porque su diseño está hecho de tal forma que se permite una sencilla ampliación por parte de terceros.
- **Entorno de desarrollo integrado** (IDE): porque proporciona herramientas para gestionar *workspaces*, para construir, lanzar y depurar aplicaciones, y para personalizar fácilmente el entorno de trabajo de programación.
- **Plataforma**: porque por sí misma no es una aplicación terminada, pero está diseñada para poder ser ampliada indefinidamente, con herramientas cada vez más sofisticadas.

Para proporcionar las diferentes funcionalidades, Eclipse se basa en mecanismos de integración y ejecución de los módulos, más conocidos como *plugins*. Para colaborar y conseguir realizar una tarea determinada, se instalan generalmente múltiples *plugins* en una misma instancia de Eclipse [28].

Algunas de las principales ventajas, y por consiguiente, razones de la elección de Eclipse a la hora de implementar la aplicación para Android, son [17]:

- Eclipse es uno de los más completos IDEs de Java disponibles en la actualidad. Es además muy fácil de usar, por lo que requiriere poco tiempo de aprendizaje.
- El *plugin* ADT para Eclipse lanzado por la OHA, permite la creación y compilación de proyectos Android, y su ejecución y depuración mediante el emulador de Android.

5.2 Planteamiento de los problemas existentes

El problema principal a resolver es la portabilidad y adaptación de una aplicación ya existente. El programa original está escrito en lenguaje de programación Python y orientado a ser utilizado en un PC. La aplicación resultante debe, sin embargo, poder ejecutarse en un dispositivo móvil basado en tecnología Android.

Los principales problemas que surgen se presentan en la Tabla 5. 1. Dichas ideas son los puntos básicos que se han tenido en cuenta a lo largo de la implementación.

Problema	Descripción
Lenguaje Python en plataforma Android	El lenguaje de programación sobre el que se desarrollan las aplicaciones en Android, es Java. Por tanto, se debe buscar un mecanismo de adaptación para poder ejecutar la aplicación escrita en Python. Se han de tener en cuenta las limitaciones que presente la herramienta y las necesidades específicas que requiera el programa original.
Gestión de ficheros	Se ha de tratar con una serie de ficheros a lo largo de la aplicación. Teniendo en cuenta entonces, la necesidad de tener almacenados dichos ficheros en el dispositivo y su manipulación por parte del programa.
Modificación de la introducción de datos de entrada	Originalmente se introducían por la línea de comandos un conjunto de datos necesarios para el funcionamiento del programa. Se ha de modificar el acceso de estos parámetros adaptándolo al entorno del terminal.
Adaptación de bibliotecas utilizadas originalmente	Verificación de la posibilidad de uso en Android, de las bibliotecas utilizadas originalmente. Estudio de las complicaciones que puedan surgir y de las medidas que se necesiten tomar para conseguir disponer de dichas bibliotecas.
Interfaz Gráfica de Usuario	Modificación de la interfaz gráfica de usuario de la aplicación. Realización de una nueva GUI propia de Android adaptada a la nueva plataforma teniendo en cuenta las nuevas dimensiones de la pantalla del dispositivo.
Obtención de imágenes	Adaptación del proceso de captación de imágenes con la cámara. Implementación y gestión de la cámara del dispositivo Android.

Tabla 5. 1: Problemas básicos a implementar

A partir de estas cuestiones se derivan otros problemas, que tienen que resolverse para poder llevar a cabo las primeras. La decisiones de cómo se han solventado todos estas dificultades se presentan en el apartado 5.3, en el que se explica tanto las soluciones a los problemas presentados en la Tabla 5. 1, como al resto de inconvenientes provenientes de los mismos.

5.3 Decisiones de implementación

A continuación se van a presentar las decisiones de implementación que se han tomado a lo largo de la realización de este proyecto, explicando con detenimiento, en función de cada punto concreto: la problemática en cuestión, las opciones existentes, complicaciones que se han ido planteando y decisiones finales.

Los ocho subapartados que se van a desarrollar a continuación, están directamente relacionados con los seis problemas presentados en la Tabla 5. 1 y los dos complementarios que se derivan de éstos.

5.3.1 Código Python sobre plataforma Android

Mediante el estudio realizado de las opciones disponibles en la actualidad para llevar a cabo el objetivo primordial de este proyecto, hacer funcionar un programa escrito en Python sobre un terminal móvil con tecnología Android, se encontró la aplicación SL4A (ver apartado 2.5). Con esta herramienta y el intérprete correspondiente Py4A (*Python for Android*), se dispone de la posibilidad de ejecutar código Python en el dispositivo Android.

'Python for Android' es el intérprete que permite instalar Python en el dispositivo mediante SL4A. En concreto, la versión utilizada de este lenguaje de programación es Python 2.6.2. Dicho intérprete (Py4A) es una aplicación Android que es necesario instalar en el terminal móvil (ver apartado 5.3.4). Es decir, el archivo `PythonForAndroid_r6.apk` del intérprete Py4A es indispensable para la ejecución de la aplicación EyeGrade, ya que posibilita la utilización de ficheros Python en Android.

Sin embargo, al pretender realizar un proyecto en su totalidad, y no un conjunto de módulos sueltos de Python, la simple instalación de la herramienta *Scripting Layer for Android* no es suficiente. Para conseguir crear el proyecto Android en función de código Python, se utilizó un fichero disponible en la página *web* del proyecto SL4A, y que sirve como plantilla para tal fin. El archivo, con nombre `script_for_android_template.zip`, está disponible en la siguiente dirección:

http://code.google.com/p/android-scripting/wiki/SharingScripts

Con él se dispone de la última versión de la herramienta SL4A, `sl4a_r4.apk`, pero permitiendo además crear un proyecto completo.

Este archivo *template* consigue su objetivo gracias a una serie de ficheros Java incluidos por defecto. En la Figura 5. 1 se presenta el código existente en el fichero `Script.java`, que hace que inicialmente el proyecto ejecute el módulo principal de Python de la aplicación EyeGrade. Dicho módulo, de nombre `main_menu.py`, está ubicado en el directorio `res/raw`. El fichero Java mencionado (`Script.java`) es uno más del conjunto que forman los seis archivos Java de la plantilla del proyecto: `DialogActivity.java`, `Script.java`, `ScriptActivity.java`, `ScriptApplication.java`, `ScriptService.java`, `TestFacade.java`.

```
public final static int ID = R.raw.main_menu;
```

Figura 5. 1: Código Java indicando el módulo Python a ejecutar

Dentro del directorio destinado a almacenar los recursos `res`, será en la carpeta `raw`, donde se guarden los ficheros Python (ver apartado 2.3.3.10). Como se explicó en el apartado 2.5.1 de la herramienta SL4A, la conexión que permite que estos módulos se entiendan con el sistema Android, se realiza a través de las líneas de código Python mostradas en la Figura 5. 2. La aplicación entera EyeGrade consta de diez módulos de Python, unos correspondientes con la tarea de creación de ficheros de examen, otros con la visualización de las imágenes de exámenes y otros con ambas funcionalidades. El conjunto entero de módulos Python que componen EyeGrade para Android, está formado por los siguientes ficheros: `configurations.py`, `configurations_create_exam.py`, `create_exam.py`, `exammaker.py`, `examparser.py`, `eyegrade.py`, `imageproc.py`, `main_menu.py`, `options.py` y `utils.py`.

```
import android
droid=android.Android()
```

Figura 5. 2: Código Python indispensable para comunicarse con Android

De la adaptación realizada para ejecutar código Python en la plataforma Android, cabe destacar también las cuatro bibliotecas incluidas en `script_for_android_template.zip`, que permiten y facilitan la creación de proyectos Android tanto con código Java como con código Python, a través de la API de Android para los primeros, y de la API adaptada para Python sobre Android para los segundos. Estas cuatro bibliotecas básicas tienen todas ellas una extensión `.jar`, es decir, son archivos de Java JAR (*Java Archive*).

5.3.2 Ficheros vinculados a la aplicación

Los ficheros utilizados en el programa original son los que se han usado también en la aplicación para Android.

En la tarea de creación de exámenes, existen tres tipos de ficheros de datos con formatos distintos. El fichero de entrada utilizado como base del diseño de los exámenes (plantilla) y los ficheros de examen de salida que son archivos Latex. El fichero de configuración de entrada que es un fichero XML, y el archivo de salida de configuración para la visualización del examen que es un archivo de texto con extensión `.eye`, formato propio de la aplicación EyeGrade. La manipulación de todos estos tipos de ficheros se realiza íntegramente en Python a través de las bibliotecas `xml.dom.minidom` y `codecs`.

Con respecto a la otra sección de la aplicación implementada, visualización de una imagen de examen analizada, se tienen dos formatos diferentes de archivos. El primero de ellos es el mismo que ya se ha comentado para la creación de exámenes, el fichero `.eye`. El que antes era fichero de salida creado por la aplicación, en este caso, será fichero de entrada de este proceso de visualización y su gestión se realizará, al igual que en la creación, mediante código Python. El otro fichero utilizado, con formato CSV, es el utilizado para guardar la información de salida, con las respuestas e información del análisis del examen. Este fichero de datos será manipulado a través de código Java, y no Python, como lo hace el programa original, a través de la biblioteca `javacsv.jar`. Este hecho es debido a la necesidad de realizar, a partir de un punto del programa, el código mediante lenguaje de programación Java (explicación más detallada en el apartado 5.3.5).

Es necesario que todos estos ficheros de datos estén almacenados en el dispositivo Android antes de la ejecución de la aplicación EyeGrade. Al ser ficheros ajenos al sistema, éste no los reconoce e imposibilita su uso, haciendo necesaria la aplicación de Android ‘Astro File

Manager', que permite la gestión de estos ficheros. Se precisa la instalación de este gestor de archivos, `ASTRO_File_Manager_2.5.2.apk`, para poder descargar los ficheros de datos aquí mencionados (ver apartado 5.3.4).

5.3.3 Introducción de parámetros de configuración

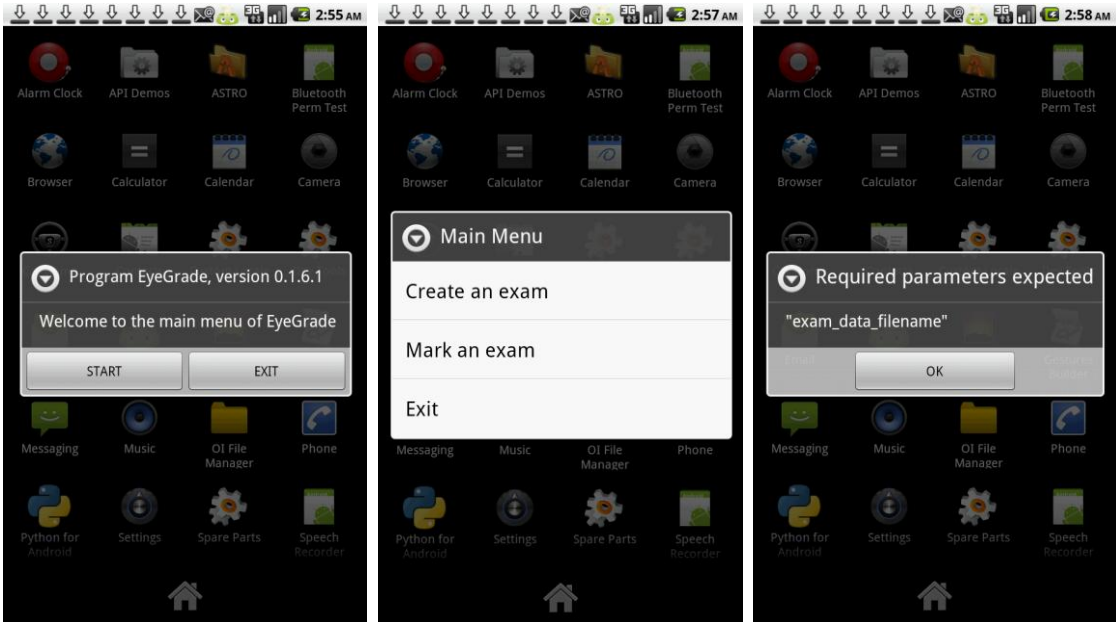
La introducción de los datos de entrada al programa original, se realiza a través de la interfaz de línea de comandos, CLI (*Command Line Interface*). Este tema es por tanto un aspecto importante a tener en cuenta a la hora de realizar la migración de la aplicación. Es necesario adaptar la entrada de parámetros de configuración a una interfaz gráfica de usuario adecuada al dispositivo móvil Android. Esta GUI se realizará mediante la sucesión de diferentes menús representados con cuadros de diálogos de Android, escritos en código Python.

Los ficheros Python que gestionan la entrada de los parámetros de configuración por parte del usuario son los siguientes: `main_menu.py`, `options.py`, `configurations_create_exam.py`, `configurations.py`. El primero de ellos es el fichero inicial de la aplicación, (el que contiene el método `main`), y el que realiza el menú principal de EyeGrade, donde el usuario decide si acceder al menú de creación de exámenes o por el contrario, realizar la visualización y análisis de los exámenes. El módulo `options.py` contiene las variables necesarias para el funcionamiento de toda la aplicación. Los valores introducidos se almacenarán en él, haciendo posible que el programa acceda a dichas variables. Los dos últimos ficheros, `configurations_create_exam.py` y `configurations.py`, corresponden con los menús de introducción de parámetros de configuración, previos al inicio de cada parte funcional de la aplicación. Con el primero de ellos se configurará e iniciará la creación de exámenes, mientras que con el segundo se procederá a realizar la corrección de los exámenes.

La biblioteca `optparse` de Python se emplea en la aplicación original para PC para realizar la obtención de los datos de entrada. En el caso de la nueva aplicación EyeGrade para Android se utiliza la biblioteca `android`, en especial la parte de la API dedicada a la interfaz de usuario. Gracias a esta colección de funciones se consigue mostrar mensajes de información en la pantalla, botones para elegir entre diversas opciones (en la Figura 5. 3 y Figura 5. 4 (a) se ve el código y una captura de pantalla de un ejemplo con dos botones), recoger la información que el usuario quiera introducir o realizar menús desplegables (ver ejemplo en la Figura 5. 5 a través del menú inicial de la aplicación EyeGrade, Figura 5. 4 (b)).

```
#Introduction to the Main Menu
def getInitMainMenu():
    droid.dialogCreateAlert('Program %s, version %s' %(utils.program_name,
       utils.version),'Welcome to the main menu of %s' %utils.program_name)
    droid.dialogSetPositiveButtonText('START')
    droid.dialogSetNegativeButtonText('EXIT')
    droid.dialogShow()
    responseInitMainMenu=droid.dialogGetResponse().result
    droid.dialogDismiss()
    if responseInitMainMenu.has_key('which'):
       resultInitMainMenu=responseInitMainMenu['which']
       if resultInitMainMenu=='positive':
              vsStop=True
       elif resultInitMainMenu=='negative':
              vsStop=False
              sys.exit(0)
    return vsStop
```

Figura 5. 3: Código de inicio de la aplicación EyeGrade



(a) Inicio de la aplicación (b) Menú principal (c) Información

Figura 5. 4: Capturas de pantallas de los código de Figura 5. 3, Figura 5. 5 y Figura 5. 6

```
# Menu of the Main Menu
def getMainMenu():
    droid.dialogCreateAlert('Main Menu')
    droid.dialogSetItems(['Create an exam','Mark an exam','Exit'])
    droid.dialogShow()
    responseMainMenu=droid.dialogGetResponse().result
    if responseMainMenu.has_key('item'):
       return responseMainMenu['item']
    else:
       return -1
```

Figura 5. 5: Código del menú principal de EyeGrade

Al orientar la aplicación al entorno móvil Android, se procede a informar al usuario de los parámetros indispensables de configuración, a través de segmentos de código como el presentado en la Figura 5. 6 cuando se está configurando la parte referente a la visualización, la funcionalidad principal de EyeGrade (ver imagen (c) de la Figura 5. 4).

```
#info required to mark the exam
def showInfoMarkExam():
    droid.dialogCreateAlert('Required parameters expected','"exam_data_filename"')
    droid.dialogSetPositiveButtonText('OK')
    droid.dialogShow()
    responseOK=droid.dialogGetResponse().result
```

Figura 5. 6: Código de información de los requisitos para la visualización de exámenes

No sólo se informa gráficamente al usuario, sino que también se realiza una comprobación de los datos introducidos. Se verifica si se han introducido los datos mínimos, y en caso de no ser así, se solicita al usuario que se proceda a su introducción. Esta comprobación, en el caso de la creación de exámenes, se puede observar a través del código de la Figura 5. 7 y Figura 5. 9, pertenecientes a las dos funciones encargadas de conseguir los datos mínimos de entrada. La Figura 5. 8 muestra el resultado de dichos métodos.

```
# Start program: create the exam
def optStart():
    droid.dialogCreateAlert('Create the Exam','Click Continue to begin the program')
    droid.dialogSetPositiveButtonText('CONTINUE')
    droid.dialogSetNegativeButtonText('BACK')
    droid.dialogShow()
    responseStart=droid.dialogGetResponse().result
    droid.dialogDismiss()
    if responseStart.has_key('which'):
        resultStart=responseStart['which']
        if resultStart=='positive':
            vStop=False
        elif resultStart=='negative':
            vStop=True
    if not vStop:
        while options.template_filename is None:
            droid.dialogCreateAlert('Paremeter missing','Template filename required')
            droid.dialogSetPositiveButtonText('OK')
            droid.dialogShow()
            responseOK=droid.dialogGetResponse().result
            options.template_filename=optTemplateFile()
        if options.exam_filename is None:
            if options.num_questions == 0 or options.num_choices == 0:
                droid.dialogCreateAlert('Paremeter missing','A file with questions,or
                                        number of questions AND choices')
                droid.dialogSetPositiveButtonText('OK')
                droid.dialogShow()
                responseOK1=droid.dialogGetResponse().result
                optFinalChoice()
                vStop=False
        else:
            if options.num_questions != 0 or options.num_choices != 0:
                droid.dialogCreateAlert('A file with questions','is mutually
                                       exclusive with number of questions and choices')
                droid.dialogSetPositiveButtonText('OK')
                droid.dialogShow()
                responseOK2=droid.dialogGetResponse().result
                optFinalChoice()
                vStop=False
    return vStop
```

Figura 5. 7: Código de verificación de parámetros

La función `optStart()` (Figura 5. 7), llamará a la función `optFinalChoice()` (Figura 5. 9), mientras el usuario no introduzca correctamente los parámetros de entrada.

Los métodos `optTemplateFile()`, `optExamFile()`, `optNumQuest()` y `optNumChoic()`, son los encargados de solicitar gráficamente, el fichero *template*, el fichero de configuración, el número de preguntas y el numero de respuestas posibles a las preguntas, respectivamente.

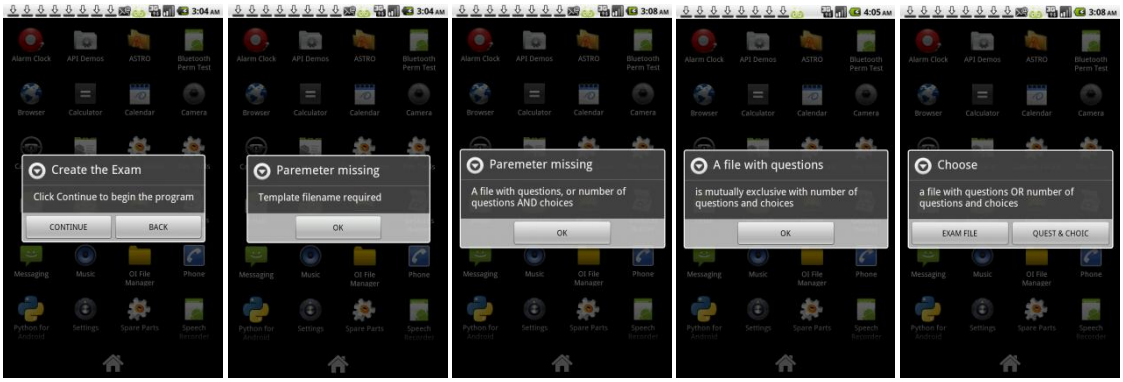


Figura 5. 8: Capturas de la verificación de parámetros

```
def optFinalChoice():
    vFinalChoiceNotDone=True
    while vFinalChoiceNotDone:
        droid.dialogCreateAlert('Choose','a file with questions OR number of
                                questions and choices')
        droid.dialogSetPositiveButtonText('EXAM FILE')
        droid.dialogSetNegativeButtonText('QUEST & CHOIC')
        droid.dialogShow()
        responseFinalChoice=droid.dialogGetResponse().result
        droid.dialogDismiss()
        if responseFinalChoice.has_key('which'):
            resultFinalChoice=responseFinalChoice['which']
            if resultFinalChoice=='positive':
                options.num_questions=0
                options.num_choices=0
                if options.exam_filename is not None:
                    vFinalChoiceNotDone=False
                 else:
                    options.exam_filename=optExamFile()
                    if options.exam_filename is not None:
                        vFinalChoiceNotDone=False
            elif resultFinalChoice=='negative':
                 options.exam_filename=None
                 if options.num_questions != 0 and options.num_choices != 0:
                     vFinalChoiceNotDone=False
                 else:
                     options.num_questions=optNumQuest()
                     options.num_choices=optNumChoic()
                     if options.num_questions != 0 and options.num_choices != 0:
                         vFinalChoiceNotDone=False
```

Figura 5. 9: Código de verificación de parámetros

Un último punto a destacar de la GUI relativa a la entrada de parámetros, es la selección tanto de ficheros como de directorios del dispositivo móvil. Originalmente se introduce la ruta del fichero o carpeta, por la línea de comandos, pero esta opción (introducir textualmente la ruta) no es operativa, ni propia de una aplicación destinada a un dispositivo móvil Android. Por tal razón, se necesita un gestor de ficheros que permita explorar los archivos y carpetas existentes dentro del terminal. No solo se necesita un administrador de ficheros, sino que se requiere manejarlo a través de código Python. Para llevar a cabo esta tarea, se ha hecho uso del programa de *OpenIntents* (OI), 'OI File Manager', instalando el archivo `FileManager-1.1.6.apk` en el dispositivo (ver apartado 5.3.4). En la Figura 5. 10 se muestra un ejemplo de cómo efectuar la elección de un fichero, en este caso, el directorio de salida donde almacenar las imágenes. Las capturas correspondientes con dicha sección de código se presentan en la Figura 5. 11.

```
# Choice the Output Dir
def optOutputDir():
    droid.dialogCreateAlert('Output Dir','store captured images at the given
                            directory')
    droid.dialogSetPositiveButtonText('OK')
    droid.dialogShow()
    responseOK=droid.dialogGetResponse().result
    map = droid.startActivityForResult( "org.openintents.action.PICK_DIRECTORY",
                      None, None, {"org.openintents.extra.TITLE":"Choose Output
                      Directory","org.openintents.extra.BUTTON_TEXT":"Choose"})
    if map.result is None:
        vOutputDir=options.output_dir
    else:
        vOutputDir = map.result["data"].replace( "file://", "" )
    return vOutputDir
```

Figura 5. 10: Código de la selección del directorio de salida

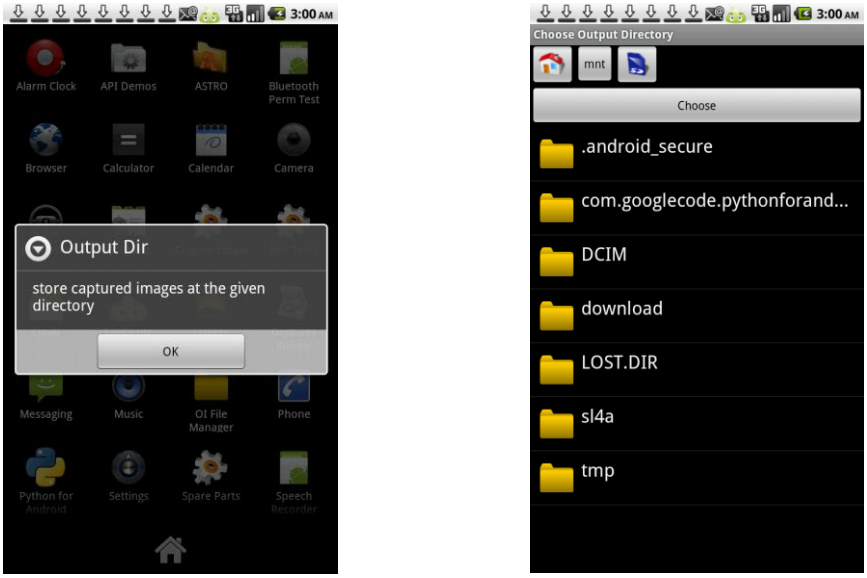


Figura 5. 11: Capturas de la selección de un directorio

5.3.4 Aplicaciones externas necesarias

Antes de ejecutar la aplicación EyeGrade, es necesaria la instalación de tres programas ajenos al proyecto implementado. Estas tres aplicaciones, como se ha explicado en los tres apartados previos, se hacen indispensables para el correcto funcionamiento de EyeGrade.

Python for Android.

Para poder ejecutar código Python en un dispositivo con tecnología Android a través del proyecto *Scripting Layer for Android,* se requiere tener instalado el intérprete Py4A. Para ello se instala el archivo `PythonForAndroid_r6.apk` en el terminal Android. Al realizar dicha acción, se instalarán las herramientas necesarias para poder disponer de la versión de Python 2.6.2 en el terminal. Las herramientas que contiene la versión 6 de Py4A son: versión 16 del intérprete, versión 14 de archivos extras de Python y versión 13 de los *scripts* de ejemplos.

Astro File Manager

Este gestor y administrador de archivos, instalado mediante el fichero `ASTRO_File_Manager_2.5.2.apk` en el dispositivo Android, es necesario para poder manipular los ficheros que la aplicación EyeGrade requiere. Mediante la versión 2.5.2 del programa 'Astro File Manager', se puede trabajar con ficheros indispensables para este proyecto pero que un dispositivo Android no soporta. Los ficheros con, por ejemplo, formatos `.xml` o Latex, son archivos de formato incompatibles con Android, y por lo tanto no se permite su descarga. Esta aplicación permite la descarga y tratamiento de dichos archivos problemáticos.

OI File Manager

La capacidad de explorar los directorios del dispositivo Android y los ficheros que en ellos se encuentran, a la hora de seleccionar los parámetros de entrada, se realiza mediante la aplicación 'OI File Manager' de *OpenIntents*. Esta acción se implementa en lenguaje de programación Python, mediante los *Intent* `PICK_FILE` y `PICK_DIRECTORY` que soporta esta aplicación. Para ejecutar dichas sentencias es necesario tener ya instalada la aplicación, mediante la versión 1.1.6 del programa, gracias al fichero `FileManager-1.1.6.apk`.

5.3.5 Migración de las bibliotecas solicitadas

Al intentar migrar la aplicación EyeGrade a la plataforma Android se ha intentado mantener el código original Python. Para alcanzar este objetivo es necesario tener en cuenta todas las variables del código inicial. Algunas de estas variables que juegan un papel importante a la hora de migrar de plataforma, son las bibliotecas. Son, de todas las bibliotecas empleadas, tres las que han conllevado mayor problema. Las bibliotecas en código Python que han provocado dicha complicación son: `opencv`, `csv` y `tre`. La primera de ellas ha forzado a realizar el programa en código Java (traducir el original), la segunda a tenido que ser sustituida por una biblioteca Java equivalente, y a la última no se le ha encontrado una posible solución.

Estas dificultades afectan a la parte del programa dedicada a la corrección de exámenes, debido a que dichas bibliotecas son empleadas en esa sección. Por tanto, el problema relativo a la adaptación de bibliotecas no influye a la hora de crear los ficheros de examen.

OpenCV

La funcionalidad principal de la aplicación, la relativa al análisis de los exámenes, se basa en el uso de la biblioteca OpenCV. OpenCV es una biblioteca de código libre orientada a la visión por computador. Es decir, proporciona herramientas para cargar, guardar y manipular imágenes.

Esta biblioteca no forma parte de las bibliotecas incluidas en la herramienta SL4A, como son por ejemplo, las bibliotecas estándar de Python (`re` o `sys`). Con la intención de reusar todo el código de la aplicación original, se intentó en un primer momento conseguir la biblioteca en Python para Android, mediante el proyecto `android-opencv`. Una vez desestimada esa opción, se realizó la búsqueda de la biblioteca OpenCV para Java. Esta opción no está aun disponible, aunque sí otra biblioteca, JavaCV, que proporciona *wrappers* de la biblioteca OpenCV original. La documentación de JavaCV, biblioteca que facilita una interfaz de OpenCV para Java en Android, y su archivo de descarga se encuentran en la página *web*:

http://code.google.com/p/javacv/

La biblioteca JavaCV hace necesario añadir al proyecto Android dos archivos de formato JAR, `javacpp.jar` y `javacv.jar`. A su vez, el uso de la biblioteca JavaCV para Android impone una versión mínima[11] del nivel API de Android. Dicho nivel mínimo es el nivel 8 (API *level* 8), o lo que es lo mismo, versión Android 2.2 más conocida como 'Froyo'.

Puesto que la funcionalidad que ofrece esta biblioteca es similar a la original, no afecta el cambio realizado entre OpenCV y JavaCV. Sin embargo, el hecho importante que implica OpenCV en el desarrollo de la implementación es la obligación de traducir todo el código Python inicial a uno equiparable en Java.

El cimentarse todo este apartado (visión artificial) en la biblioteca OpenCV hace que gran parte del código Python haya tenido que ser convertido a Java. Por tanto, una vez que se tienen todos los parámetros de configuración que haya introducido el usuario y se hayan verificado, se procederá a comenzar la actividad `MainActivity`, que se encuentra dentro del paquete de la aplicación. En la Figura 5. 12 se ve cómo se realiza esta acción. El parámetro de entrada `extras`, es el diccionario de Python que contiene los datos de configuración.

[11] JavaCV impone la condición más restrictiva a la versión Android (2.2). La única otra imposición la establecía la herramienta SL4A, funcionando con un mínimo nivel de API de valor 4 (versión 1.6 de Android). Al ser esta última menos limitativa, se necesita cumplir la más exigente.

```
myPackage="it.uc3m.eyegrade"
className='%s.MainActivity' %myPackage

action_final = droid.startActivityForResult(None, None, None,extras,myPackage,className)
```

Figura 5. 12: Código de unión entre lenguajes

En la Figura 5. 13 se pueden ver un ejemplo de la traducción del código OpenCV que se ha realizado a lo largo de este proyecto. Mostrando el código original primero (función perteneciente a la clase `ExamCapture` dentro del módulo de Python `imageproc.py`), seguido del código perteneciente a la aplicación EyeGrade para Android (en este caso el código Java se encuentra igual que antes en la clase `ExamCapture`, pero dentro del fichero `ExamCapture.java`). En este ejemplo sencillo se puede observar la similitud de las funciones en las bibliotecas.

```
def clean_drawn_image(self):
    self.image_drawn = cv.CloneImage(self.image_raw)
```

```
public void clean_drawn_image(){
    this.image_drawn=cvCloneImage(this.image_raw);
}
```

Figura 5. 13: Código ejemplo conversión OpenCV

La Figura 5. 14 muestra algunos ejemplos de instrucciones usadas en la aplicación, para importar las estructuras de la biblioteca JavaCV.

```
import com.googlecode.javacv.cpp.opencv_core.IplImage;
import com.googlecode.javacv.cpp.opencv_imgproc;
```

Figura 5. 14: Código ejemplo *imports* de JavaCV

CSV

Al haber tenido que convertir el código Python a código Java, la biblioteca relativa a los ficheros de texto con formato CSV y del mismo nombre, `csv`, ha tenido que ser modificada por una biblioteca con las mismas característica para el nuevo lenguaje. Esta biblioteca se llama `javacsv` y se compone de dos clases, `CsvReader` y `CsvWriter`. El fichero que se ha agregado en este caso al proyecto EyeGrade, es el archivo `javacsv.jar`. Las sentencias donde se puede apreciar cómo se importa dicha biblioteca, se encuentran en la Figura 5. 15 perteneciente a la clase `Exam` de Java.

```
import com.csvreader.CsvWriter;
```

Figura 5. 15: Código ejemplo *imports* de JavaCSV

Como se puede observar, solo se usa una de las dos clases, `CsvWriter`, debido a que la única función que hace uso de los ficheros `.csv`, es el método `save_answers` perteneciente a la clase `Exam`. En esta función se escriben los datos de las respuestas en el archivo `.csv`. El método, que originalmente se encontraba en el módulo `eyegrade.py`, ahora está ubicado en el archivo `Exam.java`. En la Figura 5. 16 se muestra una sección de dicha función en ambos lenguajes. En la parte superior la correspondiente al lenguaje Python y en el recuadro inferior el nuevo código de la aplicación para Android.

```
f = open(answers_file, "ab")
writer = csv.writer(f, dialect = csv_dialect)
        ...
writer.writerow(data)
f.close()
```

```
try{
   CsvWriter writer = new CsvWriter(new FileWriter(answers_file, true), ',');
        ...
   writer.write(this.studentId);
   writer.write(this.model!='-'?String.valueOf(this.model):"?");
        ...
   writer.endRecord();
   writer.close();
}catch(Exception e) {
        ...
}
```

Figura 5. 16: Código ejemplo conversión CSV

De la Figura 5. 16 se puede destacar la diferencia que existe a la hora de escribir los datos en el fichero. La variable `data` es una lista de Python con toda la información y por consiguiente, una única instrucción `writerow()` es suficiente. En Java, sin embargo, se ha de escribir cada componente con una sentencia `write()`.

TRE

La biblioteca empleada en descifrar el código numérico identificativo de los alumnos, es TRE. Dicha biblioteca (documentada en la *web* http://laurikari.net/tre/) realiza la búsqueda de expresiones regulares aproximadas.

La biblioteca, disponible para lenguaje Python, no lo está para su uso en Java. Ni esa biblioteca ni ninguna con interfaz similar. Por tanto, se ha tenido que buscar otras opciones entre las bibliotecas disponibles para el lenguaje de programación Java, que ofreciesen búsqueda de expresiones regulares aproximadas.

Existe una biblioteca un tanto similar con la que se intentó remplazar la biblioteca original, llamada FREJ (*Fuzzy Regular Expressions for Java*), pero que no dispone de los mismos recursos. Esta biblioteca tiene menos posibilidades de creación de patrones de búsqueda. Este hecho hace imposible su uso en la aplicación EyeGrade, y por tanto, es necesario encontrar otra biblioteca. Otra opción probada ha sido la biblioteca SymMetrics a través de su clase `Levenshtein` y algunas otras clases pertenecientes al paquete `simmetrics_jar_v1_6_2_d07_02_07.jar`. Pero de nuevo no se han obtenido resultados satisfactorios, puesto que no se pueden utilizar patrones de expresiones regulares, sino que la comparación se realiza entre cadenas de texto.

La API de Java contiene el paquete `java.util.regex` que está formado por las clases `Pattern` y `Matcher`. Este paquete, destinado a la manipulación de expresiones regulares y búsqueda de patrones, no es válido para el objetivo que se tiene en la aplicación EyeGrade (búsqueda aproximada), puesto que solo se obtiene una respuesta *booleana* a la búsqueda, o se cumple el patrón o no.

Por consiguiente, se ha tenido que dejar como trabajo futuro (ver apartado 7.3), la implementación de la obtención del identificador numérico del estudiante.

5.3.6 Manipulación de las estructuras de datos

Debido al uso de la nueva biblioteca JavaCV (ver apartado 5.3.5) y a la creación de la adaptada GUI en Java para Android (ver apartado 5.3.7), se tiene la necesidad de reescribir la mayor parte del código (de la sección dedicada a la visualización de imágenes) en lenguaje Java. Este cambio no solo repercute en la búsqueda y manipulación de bibliotecas equivalentes entre lenguajes, sino que también ocasiona la necesidad de analizar los tipos de datos usados en el programa.

El lenguaje de programación Python, como se puede observar en la Tabla 2. 9, ofrece una gran libertad para crear estructuras de datos de un alto nivel de complejidad. Un ejemplo de esta gran complejidad se puede observar en la Figura 5. 17, donde se presenta una parte del código perteneciente a la función `line_bounds_rank` del módulo `imageproc.py`. En ella se muestra cómo el valor devuelto por la función es una tupla compuesta por una variable de tipo *double* (`rank`) y dos puntos `p_up` y `p_down`, con sus tuplas (x,y) de tipo entero.

```
return (rank, p_up, p_down)
```

Figura 5. 17: Código original de un valor devuelto

Con los tipos de datos suministrados por el lenguaje de programación Java, no se puede realizar directamente la conversión de esta sección de código. Es decir, las restricciones impuestas por este lenguaje, al ser distintas de las limitaciones de Python, hacen que se necesiten crear estructuras de datos auxiliares, con las que poder implementar segmentos de código como el expuesto en la Figura 5. 17. Para el ejemplo mostrado en dicha figura, la estructura de datos creada es una clase a la que se le ha nombrado `RankPointEyegrade`. Esta clase tiene como atributos (como se puede comprobar en la Figura 5. 18), un atributo `double` y dos de tipo `CvPoint` (puntos de tipo `int`).

```
public class RankPointEyegrade{

        double rank;
        CvPoint point_up;
        CvPoint point_down;
        ...
}
```

Figura 5. 18: Código ejemplo de una nueva estructura de datos

El número de clases creadas para la implementación de esta aplicación asciende a un total de 8. La interconexión entre estas nuevas estructuras de datos y el resto del programa se puede ver más claramente en el diagrama de la Figura 4. 10. Estos objetos son estructuras auxiliares que están vinculadas al procesamiento de las imágenes, usadas todas ellas para llevar a cabo dicha tarea. Aparte de de la ya mencionada estructura de datos `RankPointEyegrade`, este conjunto de estructuras dedicadas al análisis de imágenes de exámenes, está formado por: `AxesEyegrade`, `CornerEyegrade`, `DigitEyegrade`, `IntegersAndPoint`, `List2Point` y su clase interna `PairPoint`. `StatsProfilerExamEyegrade` es la octava estructura de datos auxiliar (relacionada con el rendimiento de la aplicación).

```
public class DigitEyegrade{
        int decision;
        List<Double> scores;

        public DigitEyegrade(int decision,List<Double> scores){
                this.decision=decision;
                this.scores=scores;
        }
        public int getDecision(){
                return this.decision;
        }
        public List<Double> getScores(){
                return this.scores;
        }
}
```

Figura 5. 19: Código completo de una estructura de datos

La estructura interna de las clases de estos nuevos objetos no es muy compleja. Los métodos desarrollados en ellas son simples constructores, o bien métodos para obtener o modificar algún atributo o característica de éstos. Un ejemplo de estos métodos se encuentra en la Figura 5. 19, en la que se presenta la implementación de una estructura de datos completa. De estos nuevos tipos de datos de la aplicación EyeGrade, es en sus atributos donde reside su peculiaridad más importante. La Figura 5. 20 presenta algunos ejemplos de los atributos de estos nuevos tipos de datos, y la Figura 5. 21 el ejemplo concreto de la clase `StatsProfilerExamEyegrade`, estructura de datos nueva con el máximo número de atributos.

```
public class IntegersAndPoint{
    int match;
    int y_point;
    CvPoint point;
```

```
public class CornerEyegrade{
    List<List<CvPoint>> listCorner;
```

```
public class AxesEyegrade{
    float theta;
    List<CvPoint2D32f> axes;
```

Figura 5. 20: Códigos ejemplos de atributos de las nuevas estructuras de datos

```
public class StatsProfilerExamEyegrade{
    long time;
    int manual_changes;
    int num_capture;
    int num_student_id_changes;
    int id_ocr_digit_total;
    int id_ocr_digit_error;
    String id ocr detected;
```

Figura 5. 21: Código inicio `StatsProfilerExamEyegrade`

Para concluir este apartado, se va a exponer un segmento del código perteneciente tanto al programa inicial en código Python para PC, como a su equivalente Java para la aplicación Android. Esta sección presentada en la Figura 5. 22, es un método usado en el procesado de la imagen, y con él se puede comprobar cómo el código resultante (parte inferior de la imagen) es menos compacto que el original (código situado en la parte superior de la figura).

```
def id_horizontal_lines(hlines, vlines, boxes_dim):
    discard = 1 + max([box[1] for box in boxes_dim])
    lines = hlines[:-discard]
    if len(lines) < 2:
        return []
    if len(lines) > 2:
        threshold = float(min_rho_difference(hlines[-discard:])) / 2.5
        lines = collapse_lines_threshold(lines, threshold)
    lines = lines[-2:]
    if lines[1][0] - lines[0][0] < param_id_boxes_min_height:
        lines = []
    return lines
```

```
private List<CvPoint2D32f> id_horizontal_lines(List<CvPoint2D32f> hlines,
                                    List<CvPoint2D32f> vlines,int[][] boxes_dim){
        int maximBox1=boxes_dim[0][1];
        for(int i=1;i<boxes_dim.length;i++){
                if(boxes_dim[i][1]>maximBox1) maximBox1=boxes_dim[i][1];
        }
        int discard=1+maximBox1;
        List<CvPoint2D32f> lines =hlines.subList(0, hlines.size()-discard);
        if(lines.size()<2) return new ArrayList<CvPoint2D32f>();
        if(lines.size()>2){
                List<CvPoint2D32f> aux = new ArrayList<CvPoint2D32f>();
                for(int i=hlines.size()-discard;i<hlines.size();i++){
                        aux.add(hlines.get(i));
                }
                float threshold=(float)(((float)this.min_rho_difference(aux))/2.5);
                lines=this.collapse_lines_threshold(lines, threshold);
        }
        lines=lines.subList(lines.size()-2, lines.size());
        if(lines.get(1).x()-lines.get(0).x()<param_id_boxes_min_height)
                return new ArrayList<CvPoint2D32f>();
        return lines;
 }
```

Figura 5. 22: Código comparativo entre lenguajes

5.3.7 Interfaz gráfica de usuario

En el apartado 5.3.3 se ha expuesto la GUI realizada, mediante lenguaje Python, para tomar del usuario la información de entrada necesaria para la aplicación. Por tanto, este apartado se va a centrar únicamente en la interfaz gráfica de usuario relativa a la visualización de las imágenes de examen.

Ambas interfaces gráficas no solo distan en los lenguajes con los que han sido implementadas, sino que difieren en la disposición y proporciones de la pantalla. Los menús de entrada de parámetros iniciales (implementados en Python) se han diseñado con una orientación de la pantalla vertical, mientras que al mostrar la captura de las imágenes procesadas, se hará de manera horizontal.

La orientación horizontal es debida a que el programa original, y por consiguiente, la aplicación para Android, trabaja con un tamaño de imagen 640x480, y si se situase la pantalla verticalmente la imagen quedaría muy reducida. Por tanto, y como Android permite ambas orientaciones, se ha decidido escoger la opción horizontal como se puede ver en la Figura 5. 23.

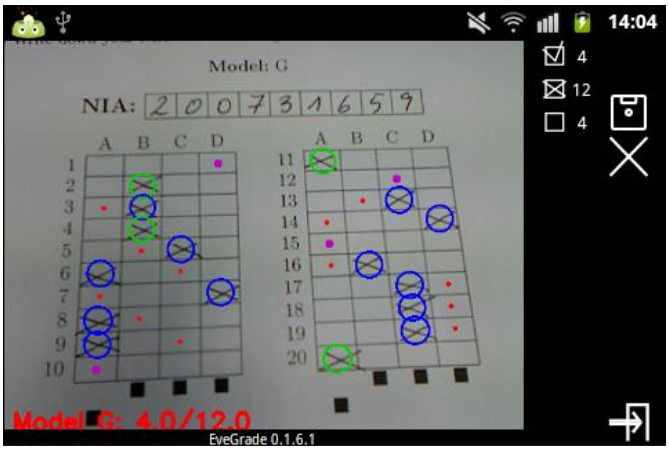


Figura 5. 23: Captura GUI

El estilo y diseño gráfico de la GUI de EyeGrade ya estaba establecido por la aplicación original. Es decir, al migrar la aplicación a la plataforma Android se ha mantenido la estructura inicial del programa. Esta estructura se basa en tres grandes bloques: una pantalla principal donde se muestra la imagen de los exámenes, unos campos de texto situados en la parte inferior, y una columna a la derecha formada por una serie de botones.

Al tratarse de un dispositivo móvil, y por tanto, disponer de una pantalla considerablemente menor que la que posee un PC, se han tenido que realizar algunos cambios con respecto a la disposición inicial de la aplicación. Se ha realizado el diseño teniendo en cuenta un tamaño estándar de pantalla de 320x480, tamaño que poseen la gran mayoría de los dispositivos Android en la actualidad. Los terminales con resolución distinta a ésta poseen una de valor superior, con lo que la aplicación EyeGrade también funcionaria correctamente en dichos dispositivos.

Teniendo en cuenta el tamaño total de la pantalla (320x480) se debe comenzar escalando la imagen del examen. El motivo de tal acción reside en el hecho de que el programa original mostraba una imagen de 640x480, valores que, en el actual dispositivo móvil, se salen del rango de la pantalla. La imagen (a) de la Figura 5. 24 muestra cómo quedaría la imagen escalada, manteniendo el resto del *layout* fiel al programa original. Se puede observar cómo, para conseguir salvaguardar la imagen proporcional en ambos ejes, queda una parte extra en la derecha de la pantalla, lo que muestra que el tamaño de la imagen capturada queda limitado por el eje vertical.

La captura, aunque de dimensión inferior, se puede ver correctamente. Sin embargo, se desperdicia mucha superficie de pantalla, dificultando por ejemplo la interacción táctil con el programa.

Para lograr aumentar las dimensiones de la pantalla principal, se trasladaron todos los campos al lado derecho de la pantalla, dejando de esta manera, agrandar verticalmente la imagen. Como se puede observar en la imagen (b) de la Figura 5. 24, el tamaño óptimo de la pantalla, se produce con un factor de escala 1.7, conseguido al dejar en la parte inferior el campo de texto informativo y en la parte de la derecha de la pantalla, la sección de la información correspondiente al análisis realizado.

El cambio estructural realizado en la interfaz gráfica de usuario (diferencia que se apreciada entre las imágenes (a) y (b) de la Figura 5. 24) permite sacar el máximo rendimiento a la superficie de la pantalla del terminal móvil.

(a) Información debajo del examen (b) Información a la derecha del examen

Figura 5. 24: Capturas de los dos diferentes *layouts*

Por tanto, el diseño final del *layout* de la aplicación EyeGrade no es exactamente similar al ideado para ordenador. Esta variación en el diseño se debe a que la GUI se ha visto obligada a adaptarse a las dimensiones (más reducidas) de un terminal móvil. Sin embargo, aún sin la misma distribución, muestra la misma información que la original.

La implementación de esta interfaz gráfica se ha realizado mediante el lenguaje de programación Java. La razón de esta decisión se debe a la no existencia de herramientas suficientes para su implementación en Python sobre Android. Dicho en otras palabras, la herramienta SL4A no permite realizar la GUI requerida por EyeGrade con lenguaje Python.

El hecho de no poder conservar el código original del programa escrito en Python y haberlo tenido que cambiar a Java ha supuesto ciertas modificaciones. Una de las más notables variaciones que se han realizado a la hora de migrar el programa está relacionada con la forma de mostrar y controlar la funcionalidad de los botones. La aplicación resultante gestiona dichos elementos mediante métodos independientes (ver un ejemplo en la Figura 5. 25 correspondiente al botón dedicado a guardar la información de salida), situados en el módulo `MainActivity.java`.

```
public void saveClick(View view){
        StatsProfilerExamEyegrade stats=this.profiler.finish_exam(this.exam);
        this.exam.save_image(this.save_pattern);
        this.exam.save_answers(this.answers_file, this.csv_dialect, stats);
        if(this.debug){
                this.exam.save_debug_images(this.save_pattern);
        }
        this.imId+=1;
        this.continue_waiting=false;
        Intent intentBeforeElseSecond = new Intent(this, DialogEyegradeActivity.class);
        startActivityForResult(intentBeforeElseSecond,this.CODE_ELSE_SECOND);

}
```

Figura 5. 25: Código ejemplo del método de un botón

Para hacerlos operativos, estas funciones son referenciadas en el archivo oportuno XML, de nombre `eyegrade.xml`, correspondiente al *layout* de la interfaz gráfica de usuario de esta sección de la aplicación (ver Figura 5. 26). La Figura 5. 25 y la Figura 5. 26 muestran la implementación del botón '*save*' de la aplicación, el cual guarda en el dispositivo la imagen procesada del examen (formato PNG, *Portable Network Graphics*) y el fichero con la información de salida.

```
<ImageButton
        android:src="@drawable/save"
        android:id="@+id/save"
                ...
        android:clickable="true"
        android:onClick="saveClick">
</ImageButton>
```

Figura 5. 26: Código ejemplo *layout* de un botón

El espíritu visual de la aplicación EyeGrade se ha conservado tanto como ha sido posible. La organización de sus principales componentes, así como las características de éstos, se han intentado preservar, a pesar de que han variado en algunos pequeños detalles. Los tres módulos estructurales de la interfaz gráfica de usuario de la aplicación para Android son:

Pantalla principal: pantalla donde se muestran las capturas realizadas, y el resultado del procesamiento de las imágenes de examen. El tamaño original de la pantalla principal destinada a mostrar el examen es 640x480, sin embargo, debido a las limitaciones que presentan los dispositivos móviles, este tamaño se ha tenido que disminuir. El tamaño mínimo designado es 377x283 en la nueva aplicación, pudiéndose variar según el terminal elegido. En caso de usar un dispositivo móvil que disponga de una pantalla mayor, se podría aumentar el tamaño de la imagen del examen. Para hacerlo operativo en el tamaño mínimo estándar que existe en la actualidad, 320x480, se han escogido los valores señalados para la pantalla principal, 377x283 (pantalla horizontal). El tamaño de la imagen con la que se trabaja si es del mismo tamaño que la originalmente utilizada, por lo que, cuando se presenta la imagen en la pantalla, ésta se reduce con un factor de escala y se sitúa en el centro del pantalla principal.

Campos de texto informativos: sección dedicada a mostrar información al usuario, situada debajo de la pantalla principal y en el lateral derecho de la pantalla. Originalmente constaba de dos elementos, uno de ellos de texto plano y el otro un campo que alterna entre dos distintos formatos, mostrando, según las circunstancias, un mensaje de texto plano o una serie de imágenes con cierta información de los resultados del examen. Debido a las restricciones que se tienen con la tecnología Android, se ha modificado esta disposición natural, dando lugar a la creación de un nuevo elemento. En la nueva aplicación se han creado dos componentes de texto, y uno adicional que será el encargado de mostrar las imágenes e información sobre la puntuación obtenida en el examen. El campo de texto plano que se ha mantenido intacto con respecto al programa inicial para PC sigue estando en la parte inferior de la pantalla, mientras que el campo de texto creado adicionalmente, junto con el que posee la información de la puntuación, se sitúan verticalmente en la zona derecha de la pantalla principal.

Menú de opciones: columna situada en el lado derecho compuesta por un conjunto de botones. Este bloque no solo ha variado en la forma de implementar sus elementos, sino que su acceso también se ha visto reducido. En la aplicación inicial, al estar orientada para PC, se tiene la opción de ejecutar las funciones que ofrecen los botones a través del teclado. Es decir, sin pulsar el botón, pero pulsando determinadas teclas, se podía conseguir el mismo objetivo. Al tratarse ahora de un terminal móvil, esta funcionalidad no tiene sentido, por lo que no se ha implementado esta opción. La usabilidad de la aplicación EyeGrade para un dispositivo Android queda basada en los botones presentes de su menú lateral. Un ejemplo más extremo de esta situación, tiene lugar con el modo *Debug* (modo de depuración de las imágenes). En el programa original para ordenador no existía ningún botón en la GUI para dicho modo de actuación, accediendo a él únicamente a través del teclado del ordenador, mientras que en la nueva aplicación para dispositivos Android, al tratarse de una pantalla táctil en un dispositivo móvil, se han creado dos botones para manejar dicha opción de depuración.

A la hora de mostrar la imagen en la pantalla del dispositivo Android, se ha tenido que variar por completo su implementación, realizándolo de manera completamente distinta a su forma original, en la que se utilizaba la biblioteca `pygame` de Python. En la Figura 5. 27 se pueden observar los dos métodos del programa original que llevan a cabo dicha acción. El primero de ellos pertenece al fichero `gui.py`, y el segundo al `imageproc.py`.

```
def show_capture(self, image, flip=True):
        pg_img = imageproc.cvimage_to_pygame(image)
        self.screen.blit(pg_img, (0,0))
        if flip:
            pygame.display.flip()
```

```
def cvimage_to_pygame(image):
    image_rgb = cv.CreateMat(image.height, image.width, cv.CV_8UC3)
    cv.CvtColor(image, image_rgb, cv.CV_BGR2RGB)
    return pygame.image.frombuffer(image_rgb.tostring(),cv.GetSize(image_rgb), 'RGB')
```

Figura 5. 27: Código Python para mostrar las imágenes

La Figura 5. 28 corresponde al código Java utilizado por la aplicación final EyeGrade, equivalente al presentado en la Figura 5. 27. Para poder mostrar correctamente la captura se ha tenido que adaptar correctamente los parámetros de configuración (8 *bits* y cuatro canales), escalar la imagen para adaptarse al tamaño inferior de la pantalla y convertirla al formato `Bitmap`, necesario para visualizarla adecuadamente.

```
public void show_capture(IplImage image){
   //image_rgb
   CvMat image_rgb=opencv_core.cvCreateMat(image.height(), image.width(), CV_8UC4);
   opencv_imgproc.cvCvtColor(image, image_rgb, CV_BGR2RGBA);
   //image_aux
   IplImage image_aux=IplImage.create(image_rgb.cvSize(),
                              opencv_core.IPL_DEPTH_8U, 4);
   opencv_core.cvSetData(image_aux, image_rgb.data_ptr(),
                              image_rgb.cvSize().width()*4);
   //image_2
   IplImage image_2= IplImage.create(Double.valueOf(image_aux.width()/
                              MainActivity.this.param_scalar_image).intValue(),
                              Double.valueOf(image_aux.height()/
                              MainActivity.this.param_scalar_image).intValue(),
                              opencv_core.IPL_DEPTH_8U, 4);
   opencv_imgproc.cvResize(image_aux, image_2, opencv_imgproc.CV_INTER_LINEAR);
   //myBitmap
   Bitmap myBitmap=Bitmap.createBitmap(image_2.width(),image_2.height(),
                              Bitmap.Config.ARGB_8888);
   myBitmap.copyPixelsFromBuffer(image_2.getByteBuffer());
   this.imageCaptured.setImageBitmap(myBitmap);
}
```

Figura 5. 28: Código Java para mostrar las imágenes

5.3.8 Captura de imágenes

Una de las partes fundamentales de la aplicación EyeGrade es la captura de imágenes a través de una cámara. Esta funcionalidad es importante para el análisis de exámenes ya contestados por los alumnos.

La toma de imágenes de los exámenes es sumamente importante, ya que el programa necesita dichas capturas para realizar el procesado de las mismas. Cierto es que se puede llevar a cabo la ejecución de la aplicación mediante el uso de imágenes ya almacenadas en el dispositivo, gracias a una de las opciones del menú de configuración. Sin embargo, y para completar la finalidad del programa original lo máximo posible, se ha hecho uso de la cámara del dispositivo Android para llevar a cabo esta tarea.

En el programa original, esta acción se desarrolla a través de una cámara *web* y mediante el control de la biblioteca OpenCV. En este caso, el control de la cámara no es posible realizarlo mediante JavaCV, y es por eso, que se han tenido que buscar otras vías alternativas de implementación. La cámara de los dispositivos Android ya viene incluida en los propios terminales, y su gestión se puede realizar de diversas maneras.

Una vez descartada la gestión a través de la biblioteca JavaCV, se intentó llevar a cabo dicha gestión a través del *Intent* propio de Android, `ACTION_IMAGE_CAPTURE`, dedicado específicamente para tal acción. Sin embargo, se decidió que ésta no era la mejor opción para realizar la función que necesita la aplicación EyeGrade por requerir la pulsación de botones por parte del usuario (la aplicación necesita realizar capturas de forma automática). La manera más adecuada para la aplicación que aquí se estudia, es la implementación del código directamente.

```
def init_camera(self, camera_id):
  self.camera = None
  if camera_id != -1:
     self.camera = self.__try_camera(camera_id)
     self.camera_id = camera_id
  if self.camera is None:
     self.camera, self.camera_id = self.__try_next_camera(-1)
  return self.camera is not None

def __try_camera(self, camera_id):
  cam = cv.CaptureFromCAM(camera_id)
  image = cv.QueryFrame(cam)
  if image is not None:
     return cam
  else:
     return None

def __try_next_camera(self, cur_camera_id):
  camera = None
  camera_id = -1
  for i in range(cur_camera_id + 1, 10) + range(0, cur_camera_id + 1):
      print "Trying camera", i
      camera = self.__try_camera(i)
      if camera is not None:
           camera_id = i
           break
  return (camera, camera_id)
```

Figura 5. 29: Código Python inicialización de la cámara

Por consiguiente, la gestión de la cámara se realiza a través de dos nuevas clases implementadas para la aplicación. Dichas clases, que se basan en la manipulación de la cámara

(`android.hardware.Camera`) distan mucho de lo implementado en el programa original para PC. Un ejemplo de esta diferencia se puede observar en la Figura 5. 29 y Figura 5. 30, donde se puede ver la inicialización de la cámara.

En el caso de Python (Figura 5. 29), todas las funciones pertenecen a la clase `ExamCaptureContext`. Las dos sentencias más importantes, y con las que se comprueba la cámara que se va a utilizar, son: `cv.CaptureFromCAM` y `cv.QueryFrame`. En el caso de la implementación en Android (Figura 5. 30), la sentencia equivalente sería `Camera.open()`. Ambas funciones son los métodos constructores de las dos nuevas clases creadas.

```
public CameraEyegrade(){
   this.previewEyegrade= new PreviewEyegrade(MainActivity.this);
}

public PreviewEyegrade(Context context){
   super(context);
   this.holderEyegrade=getHolder();
   this.holderEyegrade.addCallback(this);
   this.contextPreviewEyegrade=context;
   try {
        this.cameraEyegrade=Camera.open();
        Camera.Parameters parameters =this.cameraEyegrade.getParameters();
        parameters.setPictureSize(640, 480);
        parameters.setFocusMode(android.hardware.Camera.Parameters.FOCUS_MODE_AUTO);
       this.cameraEyegrade.setParameters(parameters);
       this.cameraEyegrade.startPreview();
       this.cameraEyegrade.setPreviewDisplay(this.holderEyegrade);
       Intent intentBeforenewIntentSecond = new
                                    Intent(((Activity)this.contextPreviewEyegrade),
                                    DialogEyegradeActivity.class);
        startActivityForResult(intentBeforenewIntentSecond,CODE_NEW_INTENT_SECOND);
   } catch (Exception e) {
        e.printStackTrace();
        Intent intentError = new Intent(((Activity)this.contextPreviewEyegrade),
                                    DialogEyegradeActivity.class);
        intentError.putExtra("errMessage","Error: Problems with Camera");
        startActivityForResult(intentError,CODE_ANY_ERROR);
   }
}
```

Figura 5. 30: Código Java inicialización de la cámara

Las sentencias que dan lugar a la inicialización de la cámara (tanto en el programa original como en la aplicación final de Android) se presentan en la Figura 5. 31. La correspondiente a Python en la parte superior, mientras que la situada inferiormente pertenece a la desarrollada para Android en Java.

Las dos clases creadas en este proyecto destinada a la gestión de la cámara son las siguientes: `CameraEyegrade` y `PreviewEyegrade`. La primera de ellas consta de un objeto de la segunda como atributo (ver Figura 5. 32).

```
imageproc_context.init_camera(select_camera(options, config))
```

```
this.camEyegrade= new CameraEyegrade();
```

Figura 5. 31: Código de comienzo de inicialización de la cámara

```
public class CameraEyegrade {
    PreviewEyegrade previewEyegrade;
            ...
```

Figura 5. 32: Atributos de `CameraEyegrade`

La clase Java `PreviewEyegrade` contiene los atributos necesarios para el control de la cámara (ver Figura 5. 33).

```
public class PreviewEyegrade extends SurfaceView implements SurfaceHolder.Callback{
    SurfaceHolder holderEyegrade;
    public Camera cameraEyegrade;
    Context contextPreviewEyegrade;
              ...
```

Figura 5. 33: Atributos de `PreviewEyegrade`

La captura de imágenes en la aplicación adaptada para Android, se desarrolla gracias a la comunicación con *Intents* y la creación de actividades. El código exacto que lleva a cabo la toma de una fotografía en la aplicación EyeGrade (perteneciente a la clase `CameraEyegrade`), se puede ver en la Figura 5. 34 en la que se presentan la función `takePictureEyegrade`.

```
public void takePictureEyegrade(){
    this.previewEyegrade.cameraEyegrade.takePicture(null,null, jpegCallback);
}
Camera.PictureCallback jpegCallback=new Camera.PictureCallback(){
    public void onPictureTaken(byte[] in_data,Camera in_camara){
        if(in_data!=null){
             FileOutputStream outStream=null;
             try {
                 if(previewEyegrade.prepareCam()==1){
                       previewEyegrade.cameraEyegrade.release();
                       Intent intentError = new Intent(MainActivity.this,
                               DialogEyegradeActivity.class);
                       intentError.putExtra("errMessage","Error: Problems with Camera");
                       startActivityForResult(intentError,CODE_ANY_ERROR);
                 }else{
                       outStream=new FileOutputStream("mnt/sdcard/imagenCaptured.png");
                       outStream.write(in_data);
                       outStream.close();
                       Intent intentOk = new Intent(MainActivity.this,
                               DialogEyegradeActivity.class);
                       startActivityForResult(intentOk,CODE_CAPTURE_CAMERA_SECOND);
                 }
             } catch (Exception e) {
                 e.printStackTrace();
                 previewEyegrade.cameraEyegrade.release();
                 Intent intentError = new Intent(MainActivity.this,
                               DialogEyegradeActivity.class);
                 intentError.putExtra("errMessage","Error: Problems with Camera");
                 startActivityForResult(intentError,CODE_ANY_ERROR);
             }
        }else{
             Intent intentOk = new Intent(MainActivity.this,
                       DialogEyegradeActivity.class);
             startActivityForResult(intentOk,CODE_MAIN_WHILE);
        }
    }
};
```

Figura 5. 34: Código Java captura de una imagen

En la Figura 5. 35 se presenta el código Python empleado en la misma tarea que el de la Figura 5. 34 para Android, es decir, en la captura de imágenes.

```
def capture(camera, clone = False):
    image = cv.QueryFrame(camera)
    if clone:
        return cv.CloneImage(image)
    else:
        return image
```

Figura 5. 35: Código Python captura de una imagen

El último punto reseñable de este apartado se encuentra en la forma de tratar la imagen obtenida por la cámara del dispositivo. En la Tabla 5. 2 se presentan las tres principales opciones que se barajaron para llevar a cabo la implementación.

Opciones	Descripción
Gestión de la imagen	Conversión de la imagen manualmente. Conversión desde el *array* de bits hasta el formato compatible usado en la aplicación
Almacenamiento interno	Guardar y obtener la imagen en la memoria cache del dispositivo
Almacenamiento externo	Guardar y obtener la imagen en la tarjeta externa *sdcard*

Tabla 5. 2: Opciones de manejo de la imagen capturada por la cámara

De las tres opciones de la tabla, que se desarrollaron y pusieron en práctica, se decidió optar por la última de ellas, guardarla en un fichero en la memoria externa. La razón primordial para tomar dicha decisión fue la no existencia de retardo entre las diferentes opciones. Al no poder optimizar esta acción mediante su almacenamiento interno o la propia gestión de la imagen, y por tanto, emplear un tiempo similar en todas ellas, se decidió seguir la misma línea de ejecución usada a lo largo de toda la aplicación, es decir, utilización de ficheros externos.

La Figura 5. 36 muestra la sentencia necesaria para obtener la imagen capturada por la cámara, y a su vez necesaria para el posterior procesamiento de la misma. Dicho código se corresponde con la captura realizada en la Figura 5. 34.

```
this.image_raw=this.load_image("mnt/sdcard/imagenCaptured.png");
```

Figura 5. 36: Código para la carga de la imagen capturada

Un ejemplo de la captura de imágenes por parte de la aplicación EyeGrade se presenta en la Figura 5. 37.

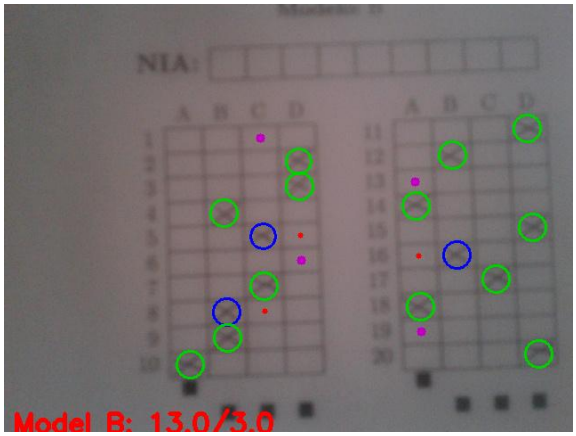


Figura 5. 37: Ejemplo de una captura de un examen con la cámara de Android

5.4 Conclusiones

Al llevar a cabo la implementación de la aplicación EyeGrade sobre la plataforma Android basada en el programa original creado para PC, se han tenido que tomar una serie decisiones necesarias para lograr realizar la migración del programa.

Para desarrollar una aplicación Android escrita en lenguaje Python se ha hecho uso de la herramienta *Scripting Layer for Android*.

Se ha modificado la forma de introducir los parámetros de configuración, adaptándolos a las características que presentan los dispositivos móviles.

La biblioteca OpenCV, encargada de la visión artificial, ha forzado a realizar parte de la aplicación mediante el uso del lenguaje Java. Hecho que a su vez ha impuesto una versión mínima 2.2 del dispositivo Android.

Se ha realizado una modificación en la forma de generar la interfaz gráfica de usuario con respecto al programa original. Adaptando también algunos pequeños detalles debido a la disminución del tamaño de la pantalla.

Se ha tenido que llevar a cabo una nueva forma de obtener imágenes a través de la cámara. Al no ser posible hacer las capturas mediante la biblioteca OpenCV, se ha realizado a través de la gestión del *hardware* de la cámara de Android.

Capítulo 6

Pruebas

6.1 Introducción

En este capítulo se comprobará el funcionamiento de la aplicación implementada, presentando las pruebas realizadas, los resultados obtenidos y las conclusiones que de éstos se adquieren.

Toda la batería de pruebas que se va a presentar en este capítulo se corresponde con la parte fundamental de la aplicación, la referente a la corrección de exámenes mediante la cámara del dispositivo móvil Android.

Desde el inicio del proyecto se hizo uso de los dispositivos virtuales que el entorno de desarrollo elegido (Eclipse) ofrece a los desarrolladores. Los AVD han sido utilizados para poder implementar y verificar la sección dedicada a la creación de los ficheros de exámenes y al correcto funcionamiento del análisis de exámenes mediante imágenes fijas almacenadas en el dispositivo. Sin embargo, las pruebas iniciales llevadas a cabo con el terminal virtual Android no se presentan en este capítulo al disponer de pruebas más completas realizadas con un terminal físico. Dicho dispositivo permite comprobar el correcto funcionamiento de la cámara a la hora de capturar las imágenes de los exámenes, realizar el posterior análisis y mostrar por pantalla los resultados.

El terminal físico empleado para llevar a cabo la verificación de este proyecto fin de carrera es un 'Samsung Galaxy Ace' con la versión 2.3.4. Al ser necesario un dispositivo Android con versión 2.2, el terminal usado cumple los requisitos mínimos ya que éste dispone de una versión superior, siendo dicha versión compatible con las versiones anteriores según el programa de compatibilidades de Android [14].

En la Figura 6. 19 se muestra un ejemplo del entorno de pruebas formado por la hoja del examen y el dispositivo físico Android. A lo largo de este capítulo sin embargo, para disponer de imágenes que permitan distinguir más claramente los resultados obtenidos se van a utilizar directamente las capturas de pantalla obtenidas mediante la herramienta SDK de Android DDMS (*Dalvik Debug Monitor Server*) (ver apartado 2.3.3.11), presentando así las imágenes capturadas de la pantalla del dispositivo físico Android.

En primer lugar se presenta el entorno de pruebas, en segundo lugar los resultados obtenidos y para terminar se presentan las conclusiones del capítulo.

6.2 Entorno de pruebas

El entorno de pruebas está formado por el dispositivo móvil Android, las distintas hojas de examen y un ordenador portátil. Este último elemento será el encargado de realizar las capturas de la pantalla del terminal, así como de obtener la información de los tiempos de ejecución de la aplicación.

En la Tabla 6. 1 se muestra de forma esquemática los cinco tipos de pruebas diferentes que se van a realizar.

Prueba	Descripción
Diferentes modelos de examen	Comprobación del funcionamiento en exámenes de **5**, **10**, **14** y **20** preguntas cada uno de ellos con modelos **A**, **B**, **C** y **D**.
Distintas contestaciones a un mismo modelo de examen	Comprobación del funcionamiento en exámenes de **5**, **10**, **14** y **20** preguntas con un modelo igual en cada uno de ellos contestado de diferente manera.
Exámenes con un elevado número de preguntas	Comprobación del funcionamiento en exámenes de **36**, **45**, **60** y **72** preguntas.
Imágenes almacenadas	Comprobación del funcionamiento en exámenes provenientes de imágenes **RAW** y **PROC** almacenadas en el dispositivo.
Modo *Debug*	Comprobación del modo depuración en sus tres variantes ***Lines***, ***Proc*** y **ambos juntos**.

Tabla 6. 1: Pruebas realizadas

A continuación se van a explicar los cinco grupos de experimentos presentados en la Tabla 6. 1 que se han llevado a cabo:

(1) **Diferentes modelos de examen**: comprobación de la posibilidad de corregir exámenes de diferentes modelos con un mismo fichero de configuración, en concreto los cuatro modelos empleados son A, B, C y D. Esta verificación se va a llevar a cabo con cuatro tipos de exámenes cada uno de ellos con un número distinto de preguntas, en concreto 5, 10, 14 y 20 son los valores empleados en esta prueba.

(2) **Distintas contestaciones a un mismo modelo de examen**: verificación de la posibilidad de corregir exámenes de un mismo modelo contestados por alumnos diferentes. De nuevo se va a realizar la prueba en exámenes con diferentes números de preguntas.

(3) **Exámenes con un elevado número de preguntas**: comprobación de la posibilidad de corregir exámenes de distintos números de preguntas como 36, 45, 60 y 72.

(4) **Imágenes almacenadas**: verificación de la corrección de exámenes sin el uso de la cámara, a través de imágenes (RAW y PROC) ya almacenadas en el dispositivo.

(5) **Modo *Debug***: comprobación del funcionamiento del modo depuración a la hora de realizar la visión artificial de los exámenes y su posterior análisis.

6.3 Resultados obtenidos

En esta sección se va a analizar los resultados obtenidos en las pruebas descritas en la sección 6.2.

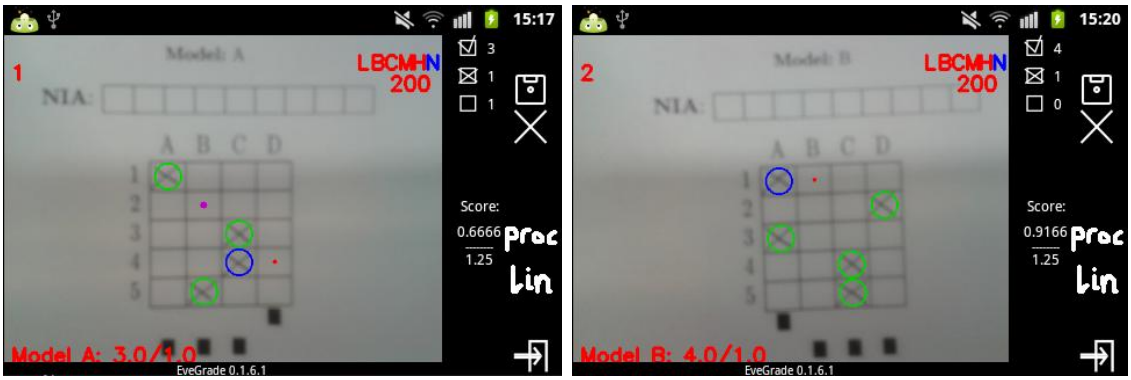
6.3.1 Diferentes modelos de examen

La prueba consiste en probar cuatro modelos de examen **A**, **B**, **C** y **D** sin reiniciar la aplicación y todos ellos con el mismo fichero de configuración. Es decir, una simulación de lo que sucede en un examen real, con las mismas preguntas para todos pero cambiadas de orden tanto las preguntas como las soluciones.

Este experimento se ha realizado con cuatro tipos de examen diferentes, cada uno con un número distinto de preguntas 5, 10, 14 y 20. A continuación se muestran los resultados obtenidos en función del número de preguntas:

6.3.1.1 Exámenes de 5 preguntas

En las Figura 6. 1 y Figura 6. 2 se pueden ver los cuatro modelos de examen correspondientes con exámenes de 5 preguntas.



(a) Modelo A de 5 preguntas (b) Modelo B de 5 preguntas

Figura 6. 1: Capturas de exámenes de 5 preguntas

(a) Modelo C de 5 preguntas (b) Modelo D de 5 preguntas

Figura 6. 2: Capturas de exámenes de 5 preguntas

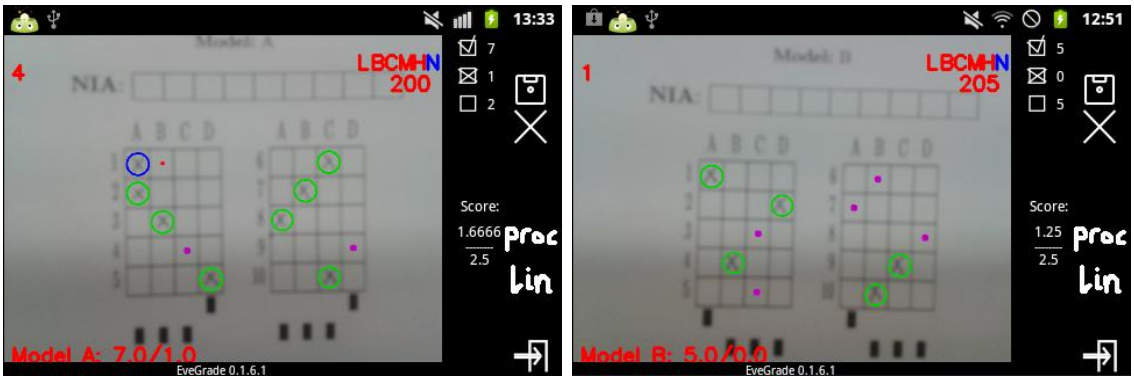
En las imágenes (a) y (b) de la Figura 6. 1 se pueden observar los modelos A y B de un examen de 5 preguntas corregido correctamente. Los modelos C y D de dicho examen se presentan en las imágenes (a) y (b) de la Figura 6. 2.

Los cuatro modelos presentados en las Figura 6. 1 y Figura 6. 2 se han analizado a partir de un mismo fichero de configuración sin necesitar reiniciar la aplicación.

Los exámenes se han analizado correctamente presentando en el menú de la derecha el número de aciertos, fallos y preguntas sin resolver.

6.3.1.2 Exámenes de 10 preguntas

En las Figura 6. 3 y Figura 6. 4 se pueden ver los cuatro modelos de examen correspondientes con exámenes de 10 preguntas.



(a) Modelo A de 10 preguntas (b) Modelo B de 10 preguntas

Figura 6. 3: Capturas de exámenes de 10 preguntas

(a) Modelo C de 10 preguntas (b) Modelo D de 10 preguntas

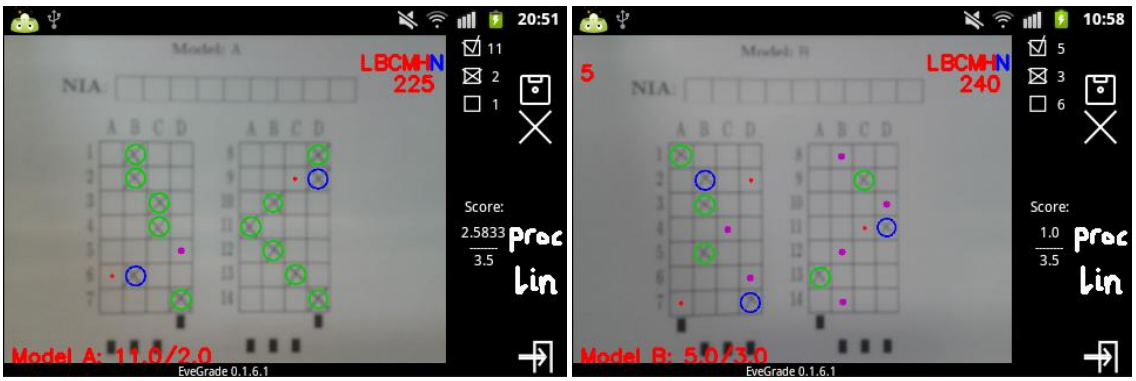
Figura 6. 4: Capturas de exámenes de 10 preguntas

Las imágenes de las Figura 6. 3 y Figura 6. 4 muestran los análisis realizados con éxito de los modelos A, B, C y D de los exámenes de 10 preguntas.

El tamaño de las casillas se ha visto disminuido debido al aumento del número de tablas. Esto se puede comprobar comparando las imágenes de la Figura 6. 1 (exámenes de 5 preguntas con una sola tabla) y las de la Figura 6. 4 (exámenes de 10 preguntas con dos tablas).

6.3.1.3 Exámenes de 14 preguntas

En las Figura 6. 5 y Figura 6. 6 se pueden ver los cuatro modelos de examen correspondientes con exámenes de 14 preguntas.



(a) Modelo A de 14 preguntas (b) Modelo B de 14 preguntas

Figura 6. 5: Capturas de exámenes de 14 preguntas

(a) Modelo C de 14 preguntas (b) Modelo D de 14 preguntas

Figura 6. 6: Capturas de exámenes de 14 preguntas

En las imágenes (a) y (b) de la Figura 6. 5 se pueden observar los modelos A y B de un examen de 14 preguntas, mientras que en las imágenes de la Figura 6. 6 se presentan los modelos C y D.

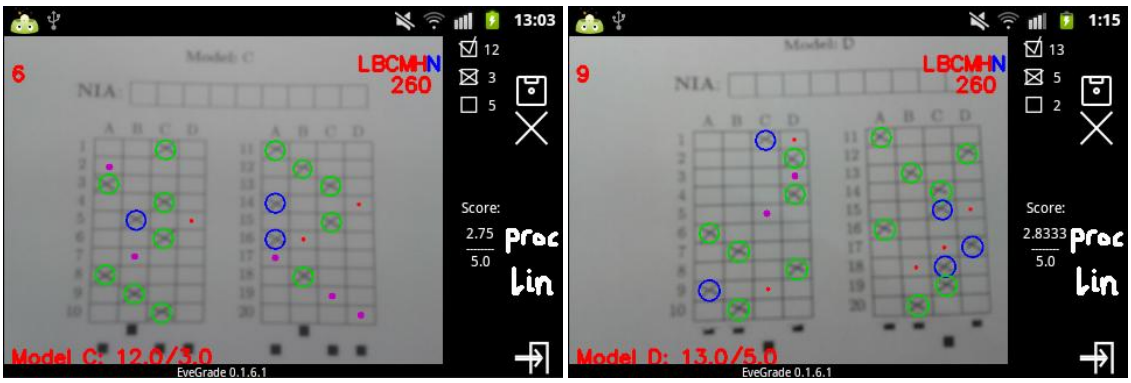
En la parte inferior de la pantalla se muestra con color rojo el modelo detectado por la aplicación y el número de aciertos y de fallos conseguidos en cada examen.

6.3.1.4 Exámenes de 20 preguntas

En las Figura 6. 7 y Figura 6. 8 se pueden ver los cuatro modelos de examen correspondientes con exámenes de 20 preguntas. En concreto, en las imágenes (a) y (b) de la Figura 6. 7 se pueden observar los dos primeros modelos (A y B), mientras que en las dos siguientes imágenes de la Figura 6. 8 se presentan los modelos C y D.

(a) Modelo A de 20 preguntas (b) Modelo B de 20 preguntas

Figura 6. 7: Capturas de exámenes de 20 preguntas



(a) Modelo C de 20 preguntas (b) Modelo D de 20 preguntas

Figura 6. 8: Capturas de exámenes de 20 preguntas

Al aumentar el número de preguntas, el examen mostrado en la pantalla del dispositivo móvil se ha visto disminuido en tamaño.

Este hecho hace que se visualicen las celdas con menor precisión que por ejemplo las presentes en exámenes de 5 preguntas como los de la Figura 6. 1. Sin embargo, la imagen de los exámenes sigue siendo correcta y la posibilidad de cambiar algún resultado mediante la pantalla táctil también sigue siendo sencilla de realizar.

6.3.2 Distintas contestaciones a un mismo modelo de examen

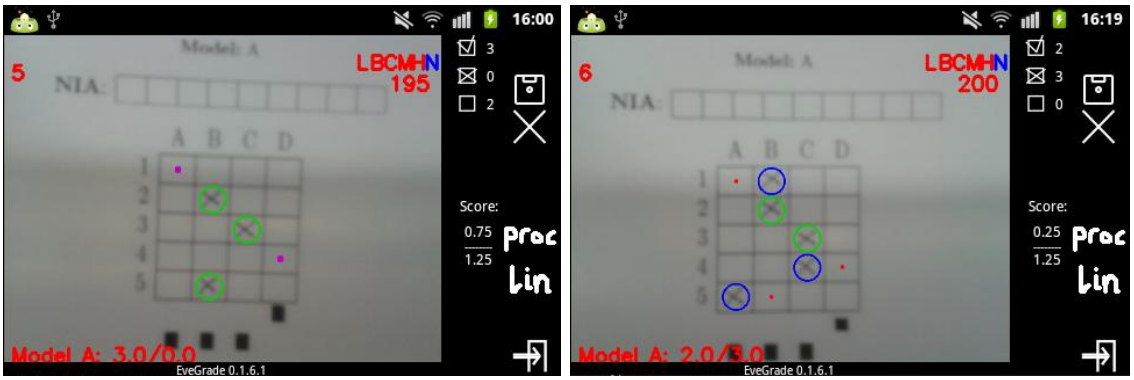
La prueba consiste en probar **un mismo modelo** de examen A, B, C ó D en dos exámenes, sin reiniciar la aplicación y con el mismo fichero de configuración. En otras palabras, una simulación de lo que sucede en un examen real cuando dos alumnos contestan diferente a un examen idéntico.

A la hora de realizar esta prueba se han empleado las capturas de pantalla del dispositivo móvil para comprobar el funcionamiento de la aplicación.

Este experimento se ha realizado con cuatro tipos de examen diferentes, cada uno con un número distinto de preguntas 5, 10, 14 y 20. A continuación se muestran los resultados obtenidos en función del número de preguntas:

6.3.2.1 Exámenes de 5 preguntas

A continuación se presentan las dos soluciones diferentes realizadas para un mismo examen de 5 preguntas, ambos correspondientes al modelo A.



(a) Modelo A de 5 preguntas (b) Modelo A de 5 preguntas

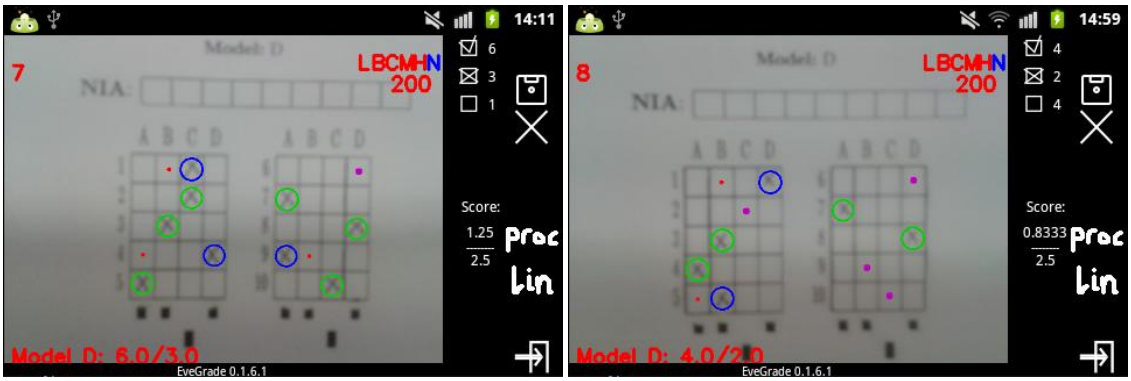
Figura 6. 9: Capturas de exámenes de 5 preguntas

La Figura 6. 9 presenta dos contestaciones distintas al mismo examen, y por tanto cada una de ellas con una calificación determinada.

Se puede ver cómo la solución correcta de cada pregunta es igual en ambos exámenes dado que ambos tienen el mismo modelo y se han corregido con el mismo fichero de configuración.

6.3.2.2 Exámenes de 10 preguntas

A continuación se presentan las dos soluciones diferentes realizadas para un mismo examen de 10 preguntas, ambos correspondientes al modelo D.



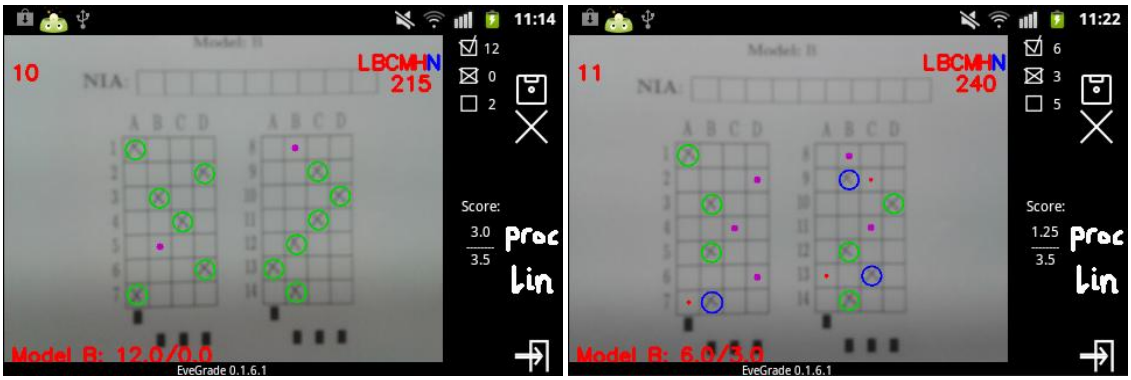
(a) Modelo D de 10 preguntas (b) Modelo D de 10 preguntas

Figura 6. 10: Capturas de exámenes de 10 preguntas

La Figura 6. 10 muestra las diferentes calificaciones obtenidas por dos alumnos al realizar un mismo examen de 10 preguntas.

6.3.2.3 Exámenes de 14 preguntas

A continuación se presentan las dos soluciones diferentes realizadas para un mismo examen de 14 preguntas, ambos correspondientes al modelo B.



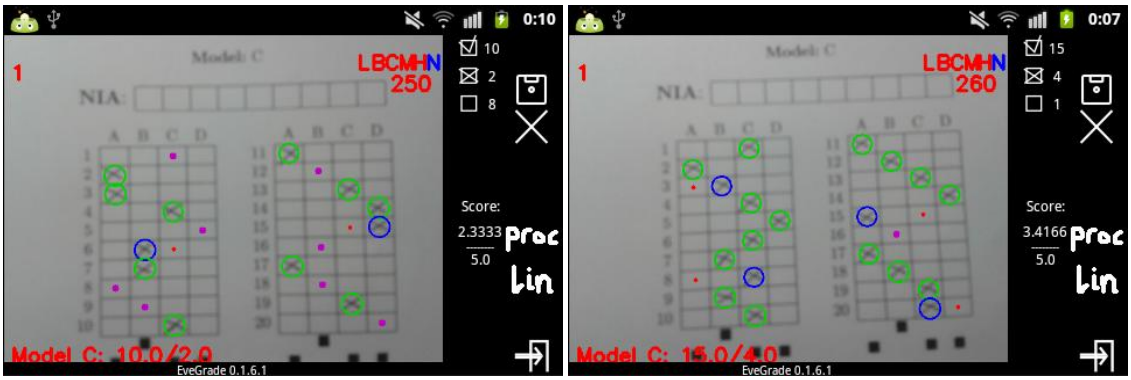
(a) Modelo B de 14 preguntas (b) Modelo B de 14 preguntas

Figura 6. 11: Capturas de exámenes de 14 preguntas

En las imágenes (a) y (b) de la Figura 6. 11 se ve el análisis realizado en dos exámenes de 14 preguntas cada uno y con las mismas soluciones pero resuelto de forma distinta.

6.3.2.4 Exámenes de 20 preguntas

A continuación se presentan las dos soluciones diferentes realizadas para un mismo examen de 20 preguntas, ambos correspondientes al modelo C.



(a) Modelo C de 20 preguntas (b) Modelo C de 20 preguntas

Figura 6. 12: Capturas de exámenes de 20 preguntas

La Figura 6. 12 muestra las diferentes calificaciones obtenidas por dos alumnos al realizar un mismo examen de 20 preguntas.

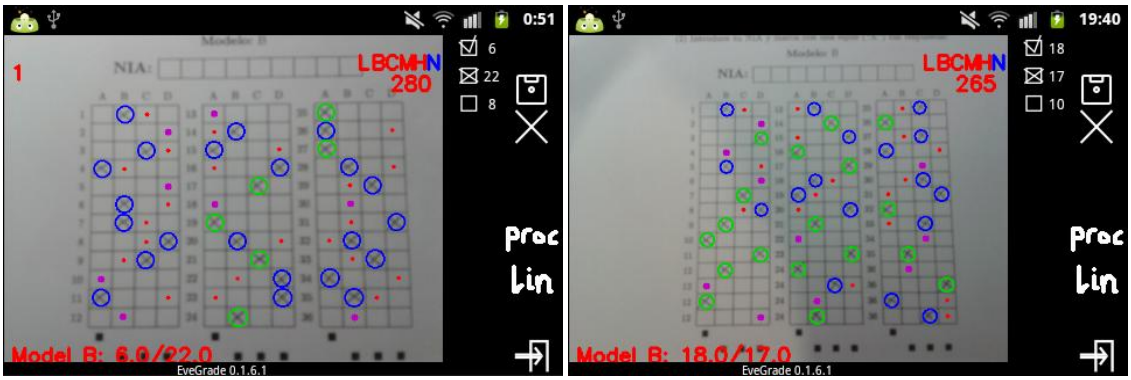
La calificación de ambos se muestra a la derecha de la imagen, en la sección donde se indica "*Score:*" y teniendo en cuenta que cada respuesta correcta suma 1/4, cada respuesta incorrecta resta 1/12 y las respuestas no contestadas ni suman ni restan.

6.3.3 Exámenes con un elevado número de preguntas

Para corroborar el correcto funcionamiento de la aplicación para corregir exámenes se ha procedido a probar la aplicación EyeGrade para Android con diferentes exámenes, todos ellos con un número elevado de preguntas. Los exámenes utilizados en este conjunto de pruebas constan de **36, 45, 60 y 72** preguntas.

Las Figura 6. 13 y Figura 6. 14 muestran dichos exámenes resueltos. En todos ellos se puede hacer uso de la funcionalidad de cambiar valores del análisis mediante la pantalla táctil pulsando en la celda deseada. Esta función de la aplicación, aunque operativa, deja de ser tan sencilla como pueda resultar con exámenes de 10 preguntas debido a la necesidad de tener más precisión al seleccionar la casilla de tamaño más reducido.

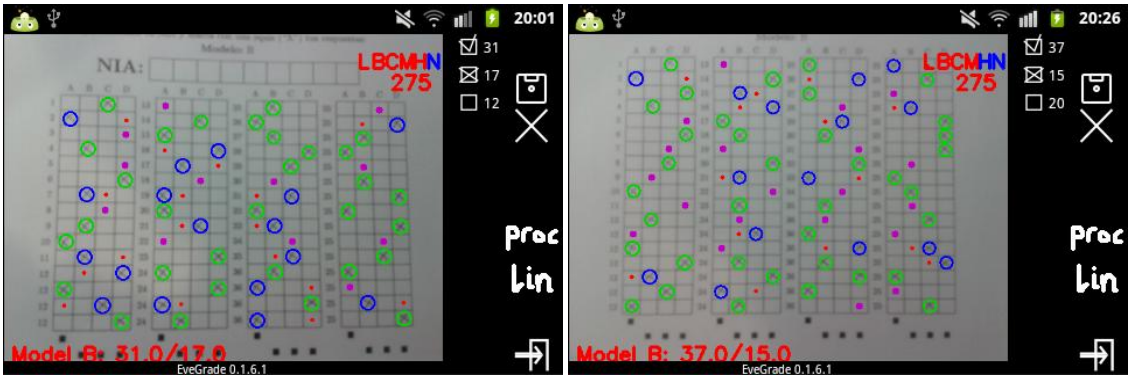
La imagen (b) de la Figura 6. 14 correspondiente con un examen de 72 preguntas con 4 posibles opciones muestra el límite aproximado del examen máximo que esta aplicación puede corregir, teniendo en cuenta que la cámara debe captar el examen completo y que se necesita un mínimo de resolución para diferenciar las tablas y las respuestas del examen.



(a) Examen de 36 preguntas (b) Examen de 45 preguntas

Figura 6. 13: Capturas de exámenes de 36 y 45 preguntas

La imagen (a) de la Figura 6. 13 muestra un examen resuelto de 36 preguntas y la imagen (b) de la Figura 6. 13 uno de 45, ambos con cuatro posibles soluciones para cada pregunta.



(a) Examen de 60 preguntas (b) Examen de 72 preguntas

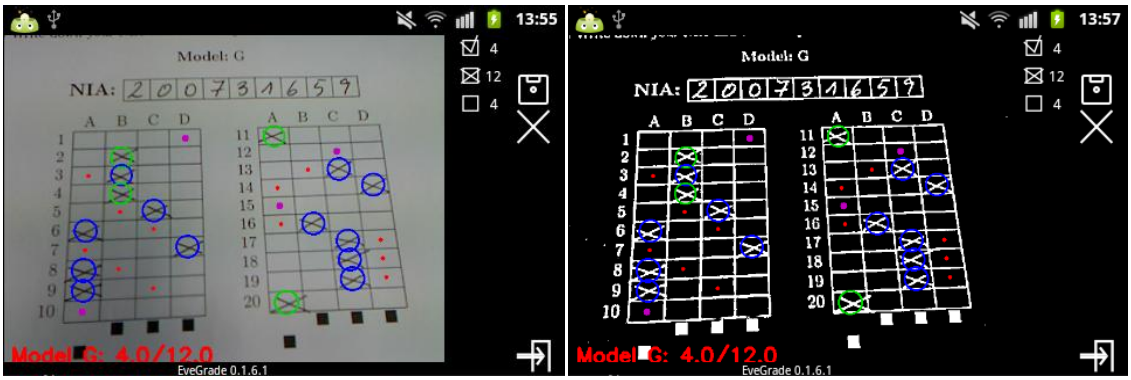
Figura 6. 14: Capturas de exámenes de 60 y 72 preguntas

La imagen (a) de la Figura 6. 14 muestra un examen resuelto de 60 preguntas y la imagen (b) de la Figura 6. 14 uno de 72.

6.3.4 Imágenes almacenadas

Además del uso de la cámara del dispositivo móvil Android, se puede realizar el análisis de imágenes de examen ya guardados en el terminal. En esta prueba se ha llevado a cabo la comprobación de dicha funcionalidad.

Las imágenes (a) y (b) de la Figura 6. 15 son un ejemplo de la correcta corrección de los exámenes cuya fuente de origen son imágenes almacenadas en el interior del dispositivo móvil. Según sean las características de la imagen (RAW (a) o PROC (b)) así se indicará el tipo de imagen a analizar en el menú de configuración.



(a) Imagen RAW (b) Imagen PROC

Figura 6. 15: Capturas de imágenes de examen almacenadas en el dispositivo

6.3.5 Modo *Debug*

El modo de funcionamiento encargado de la depuración de las imágenes es realmente útil a la hora de comprobar la veracidad de los análisis o para detectar algún problema que pueda surgir y que impida realizar correctamente la corrección del examen.

Este modo de funcionamiento posee dos opciones diferentes, cada una de ellas con un botón en la barra derecha del menú:

- ***Debug lines***: muestra las líneas verticales y horizontales detectadas por la aplicación y los puntos de unión de las tablas.
- ***Debug proc***: presenta la imagen procesada, mostrando de negro los espacios blancos y de blanco las líneas negras de la imagen.

Se van a probar dichos modos de funcionamiento (ver Figura 6. 16 y Figura 6. 18) así como la combinación de ambos (ver Figura 6. 17).

Gracias a estos modos de depuración se pueden comprobar cómo las líneas detectadas deben cumplir una serie de requisitos como el de estar situadas perpendicularmente entre ellas o el que las líneas que forman las celdas de las tablas estén proporcionalmente separadas.

Se han empleado las capturas de pantalla del dispositivo móvil Android para comprobar el funcionamiento del mencionado modo de depuración.

A continuación se presentan las capturas correspondientes con los modos: *debug lines*, *debug lines* junto con *debug proc* y *debug proc* solamente.

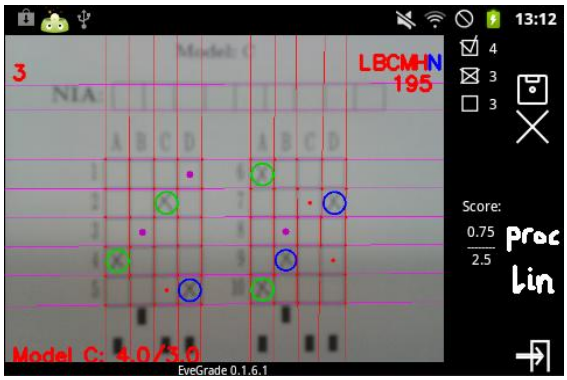


Figura 6. 16: Captura modo *debug lines*

En la Figura 6. 16 se presenta la ejecución del modo *debug lines* aplicado sobre un examen de 10 preguntas.

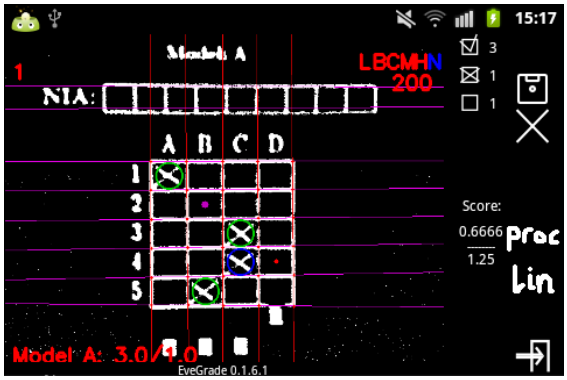


Figura 6. 17: Captura modo *debug lines* y *debug proc*

En la Figura 6. 17 se presenta la ejecución del modo *debug lines* en conjunto con el modo *debug proc* aplicado sobre un examen de 5 preguntas.

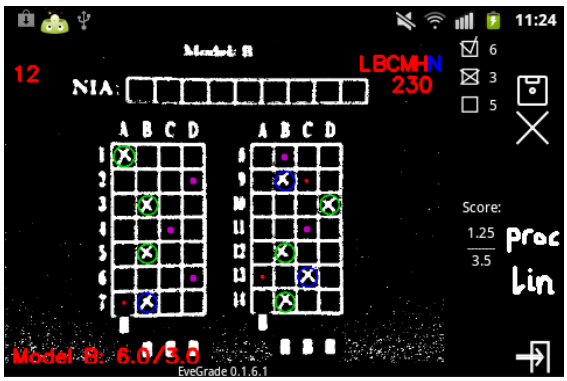


Figura 6. 18: Captura modo *debug proc*

En la Figura 6. 18 se presenta la ejecución del modo *debug proc* aplicado sobre un examen de 14 preguntas.

6.4 Conclusiones

A la vista de los resultados obtenidos tras la realización de las pruebas se puede concluir que la aplicación consigue realizar su principal función, la corregir exámenes mediante la cámara del dispositivo móvil Android.

En la Tabla 6. 2 se muestra el conjunto de pruebas realizadas en este capítulo indicando si su ejecución se ha llevado a cabo con éxito o no.

Prueba	¿Ejecutada con éxito?
1. Diferentes modelos de examen	Si
2. Distintas contestaciones a un mismo modelo de examen	Si
3. Exámenes con distintos números de preguntas	Si
4. Imágenes almacenadas	Si
5. Modo *Debug*	Si

Tabla 6. 2: Comprobación del éxito en las pruebas realizadas

Se ha comprobado el correcto funcionamiento de la aplicación con distintos e iguales modelos en exámenes de 5, 10, 14 y 20 preguntas. Se ha verificado que la aplicación soporta exámenes con un número elevado de preguntas como 36, 45, 60 y 72.

La cámara del terminal no es la única fuente posible de imágenes, sino que la aplicación funciona también con imágenes fija almacenadas previamente en el dispositivo. Se ha comprobado la funcionalidad que ofrece la aplicación a la hora de depurar las imágenes de los exámenes que se van a analizar.

El tiempo[12] empleado en el procesamiento de las imágenes es excesivo debido al alto grado de computación que requiere la función `cvHoughLines2()`. Este tiempo extra[13] es la principal desventaja de la aplicación EyeGrade para Android con respecto al mismo programa para PC. La diferencia de tiempos (desfavorable para la aplicación de Android) deja abierta una línea futura de trabajo para mejorar la velocidad del procesamiento de la imagen en Android.

Por último para concluir, se presenta la Figura 6. 19 donde se muestra una ejecución exitosa de la aplicación EyeGrade a través del dispositivo móvil Android.

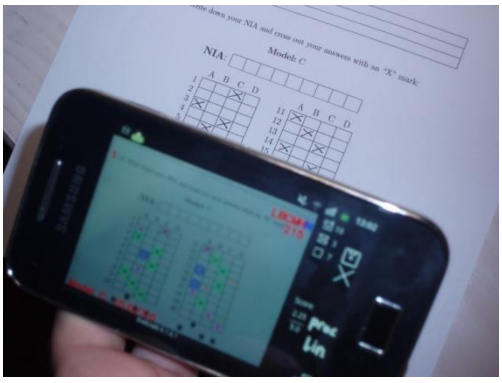


Figura 6. 19: Ejecución de la aplicación EyeGrade

[12] Los tiempos se han medido a través de la herramienta SDK de Android DDMS.

[13] La función `cvHoughLines2()` invierte una media de 13 segundos, con valores que oscilan entre los 11 y los 15 segundos, tiempos todos ellos muy superiores a los empleados en el programa original para ordenador

Capítulo 7

Conclusiones finales y trabajos futuros

7.1 Conclusiones finales

En este proyecto se ha realizado la migración de un programa en código Python para PC, a otra aplicación implementada en Java y en Python para la plataforma Android. La función principal de la aplicación es la corrección de exámenes multirespuesta mediante un sistema de visión artificial.

- Se ha logrado desarrollar por completo la aplicación mediante herramientas de *software* libre (*open source*), con lo que se consigue que sea accesible a todo el mundo y modificable por parte de terceros de forma razonable y relativamente asequible.

- Se ha implementado un sistema en el que no es necesario ningún dispositivo tecnológico externo. El dispositivo móvil con sistema operativo Android, junto con la hoja de examen, es suficiente para poder ejecutar y llevar a cabo dicha aplicación.

- Los ficheros de texto utilizados para la configuración de ambas partes de la aplicación, así como los ficheros de imagen de salida, son ficheros que ocupan poco espacio en memoria, haciendo que EyeGrade no ocupe demasiada memoria (recurso importante en los dispositivos móviles) dentro del dispositivo móvil Android.

- La aplicación es extensible gracias a la posibilidad de agregar de forma sencilla diferentes módulos que añadan nuevas funcionalidades o innovaciones a las actuales, posibilitando de esta manera, realizar nuevas actualizaciones de la aplicación.

- La creación de una interfaz gráfica de usuario adaptada a la plataforma Android, pero que mantuviese la esencia del programa original. Éste ha sido otro punto importante del proyecto, al ver las limitaciones que supone un dispositivo móvil de dimensiones reducidas y tener que adaptarse a ellas.

- La implementación mediante código Python de la aplicación en su totalidad habría sido una mejor opción en caso de haber sido posible. De haberse desarrollado todo en Python se podría actualizar la aplicación de forma sencilla con la última versión del programa para PC.

7.2 Aspectos importantes de la migración

- Uno de los grandes problemas que se ha encontrado en la realización del proyecto, fue la implementación en Python de una aplicación Android. Para llevar a cabo dicha tarea se hizo uso de la herramienta SL4A, aunque la documentación existente acerca de este tema está principalmente orientada a la ejecución de *scripts* en Python, pero no a la creación de proyectos completos desarrollados en este lenguaje.

- Los mayores problemas surgieron con la biblioteca OpenCV y su intento de funcionamiento en lenguaje Python sobre la plataforma Android. Al basarse gran parte de la aplicación en dicha biblioteca, ha sido necesario adaptarla a la nueva plataforma, inicialmente en Python para Android, y una vez que se vio su imposibilidad, en lenguaje Java para Android.

- Otro punto a destacar de este sistema ha sido la traducción del código original Python a Java. Debido a que el procesado de la imagen es el núcleo central de la aplicación se ha tenido que ser cuidadoso al transformar el código, lo que ha supuesto un trabajo delicado y complejo.

- La captura de imágenes y su posterior visualización en la pantalla ha sido un reto importante de este proyecto. La conversión entre el formato mostrado en la pantalla Android y el utilizado en el procesado de la imagen llevó un tiempo de estudio hasta la adaptación entre ambos.

- La obtención de imágenes a través de la cámara del dispositivo Android ha sido un hito significativo de la migración de la aplicación. El control del *hardware* de la cámara, así como el manejo de las imágenes capturadas por la misma, se ha tenido que implementar de manera totalmente diferente con respecto al programa original para PC.

- No se ha podido desarrollar la función de identificación del estudiante que ha realizado el examen. Se necesita encontrar una biblioteca que cumpla con los mismos requisitos que la biblioteca TRE de Python, pero en el lenguaje Java para la plataforma Android. Una biblioteca que permita una búsqueda aproximada de expresiones regulares en función de una serie de patrones de búsqueda.

- Se ha implementado todo lo referente a la función que detecta los identificadores numéricos, verificando su correcto funcionamiento y comprobando que una vez que se encuentre tal biblioteca, el resto de la aplicación está lista para llevar a cabo la función.

- La diferencia en la velocidad de procesamiento ha hecho que se note la migración realizada entre plataformas. Un inconveniente que presenta la aplicación implementada en Android es la capacidad menor que posee un dispositivo móvil con respecto a un ordenador de sobremesa.

- Un aspecto que distingue a esta aplicación del programa original ha sido la generación de los menús de inicio. Una nueva interfaz de usuario, con la que inicialmente el programa lleva a cabo la configuración de los parámetros de entrada.

7.3 Trabajos futuros

Los siguientes cinco puntos se proponen como líneas de trabajo futuro para desarrollar y conseguir una funcionalidad de la aplicación EyeGrade más completa.

Trabajo futuro	Descripción
Identificador de estudiante	Validar el identificador del estudiante mediante la visión artificial de la imagen del examen.
Tamaño variable de la GUI	Realizar una GUI variable en función del dispositivo móvil utilizado a partir de un valor mínimo de pantalla.
Optimización del procesado de imagen	Optimización del procesamiento de la imagen del examen. Consiguiendo hacer de EyeGrade una aplicación operativa en tiempo real.
Precisión en la imagen mostrada en la GUI	Seleccionar de la imagen capturada solo la sección que contiene las tablas de respuestas.
Optimización del enfoque de las capturas	Optimización de la toma de imágenes realizada con la cámara del dispositivo móvil Android.

Tabla 7. 1: Líneas de trabajo futuro

A continuación, se va a explicar más detalladamente lo expuesto en la Tabla 7. 1 con el fin de definir las posibles opciones de acción futuras.

Validar el identificador del estudiante. Se ha de completar la funcionalidad propuesta por el programa original, encontrando una biblioteca equivalente a TRE para Java sobre Android, con la que terminar de realizar la comprobación aproximada de patrones.

Se ha de llevar a cabo una búsqueda de una biblioteca que realice las mismas funciones que la usada en el programa original ó en su defecto, crearla explícitamente para el lenguaje de programación Java.

Realizar una interfaz gráfica de usuario variable en función del dispositivo móvil utilizado. La aplicación ha de dar la opción en la configuración de la aplicación de utilizar un tamaño mayor o menor, dependiendo del tamaño de la pantalla que se disponga.

Es posible añadir como parámetros de entrada la resolución de la pantalla que se va a emplear, y en función de dichos datos, crear el *layout* correspondiente a las dimensiones introducidas. Con esta nueva opción, se conseguiría aumentar el tamaño de las imágenes capturadas, no teniendo que escalar las capturas o hacerlo con un factor de escala menor.

Optimizar el procesamiento de imagen para realizarlo en un tiempo menor. Se necesita realizar la sentencia que ejecuta la instrucción `cvHoughLines2()`, en un tiempo inferior del que emplea en la actualidad.

Para lograr realizar dicho objetivo, y sin contar con la opción de disponer de un terminal con mayor capacidad de procesamiento, se disponen de dos opciones principales.

(a) Realizar el procesamiento de la imagen de forma externa al dispositivo móvil, mandando la imagen capturada a un servidor y obteniendo de él ya todo el análisis

final. De esta forma se aprovecharía el aumento de velocidad que un servidor conlleva con respecto a un dispositivo Android.

(b) Buscar alguna otra forma alternativa de realizar dicha instrucción, que es la encargada de detectar las líneas existentes en la imagen.

Seleccionar de la imagen capturada la sección con información del examen y mostrarlo por pantalla. En lugar de mostrar la imagen capturada se mostraría una sección de ésta pero ocupando el mismo espacio, con lo que se conseguiría aumentar el tamaño de la tabla mostrada en la pantalla del dispositivo móvil.

A partir de la información disponible de la imagen se realizará la selección del área con la información del examen, ajustándola al tamaño de la pantalla principal para ser mostrada, consiguiendo así facilitar la interacción táctil con el programa.

Realizar la captura de imágenes con un sistema de enfoque adaptativo. Aunque el programa es robusto y funciona correctamente con imágenes no muy nítidas, la aplicación puede mejorar este aspecto consiguiendo adaptar el enfoque de la cámara.

Se necesita encontrar alguna herramienta que permita variar el enfoque de forma adaptativa en función del entorno de la imagen sin requerir la pulsación de botones por parte del usuario.

Bibliografía

[1] BARBIER, Frédéric; LAVENIR, Catherine Bertho. “Historia de los medios de Diderot a Internet”. 1ª ed. 1º reimp. Buenos Aires: Colihue, 2007. 416p. ISBN: 978-950-581-267-7

[2] MARTÍN DE BERNARDO GONZÁLEZ, Cesar; PRIEDE BERGAMINI, Tiziana. “Marketing móvil: una nueva herramienta de comunicación”. 1ª ed. Spain: Netbiblo, 2007. ISBN: 978-84-9745-182-6

[3] Real Academia Española. http://www.rae.es [Consulta año 2011]

[4] KIFER, Michael; SMOLKA, Scott A. “Introduction to operating system design and implementation”. London: Springer, 2007. ISBN: 978-1-84628-842-5

[5] UNHELKAR, Bhuvan. “Handbook of research in mobile business: technical, methodological and social perspectives”. London: Idea Group Inc, 2006. ISBN: 1-59140-817-2

[6] Gartner PR Mobile Phones 2010. http://visualescrita.files.wordpress.com/2011/02/gartner-pr-mobile-phones-2010.pdf [Consulta año 2011]

[7] MORRIS, Ben. “The Symbian OS architecture sourcebook”. England: John Wiley & Sons, 2007. ISBN: 978-0-470-01846-0

[8] SWEENY, Alastair. “Blackberry planet”. Canada: John Wiley & Sons, 2009. ISBN: 978-0-470-15940-8

[9] ZHOU, Zhinan; ZHU, Robert; ZHENG, Pei; YANG, Baijian. “Windows Phone 7 programming for Android and iOS developers”. Indianapolis: John Wiley & Sons, 2011. ISBN: 978-1-1180-2197-2

[10] Revista Digital, Red Futura. http://www.revistaredfutura.com [Consulta año 2011]

[11] Developer Economics 2011. http://www.visionmobile.com/rsc/researchreports/VisionMobile-Developer_Economics_2011.pdf [Consulta año 2011]

[12] MEIER, Reto. “Professional Android application development”. Indianapolis: Wiley Publishing Inc, 2009. ISBN: 978-0-470-34471-2

[13] Open Handset Alliance. http://www.openhandsetalliance.com/oha_faq.html [Consulta año 2011]

[14] Android Developers. http://developer.android.com/index.html [Consulta año 2011]

[15] GARGENTA, Marko. “Learning Android”. 1ª ed. United States: O´Reilly Media Inc, 2011. ISBN: 978-1-449-39050-1

[16] BURNETE, Ed. HELLO, ANDROID. “Introducing Google´s mobile development platform”. Texas: The Pragmatic Bookshelf, 2009. ISBN: 978-1-934356-17-3

[17] DIMARZIO, J.F. “Android, a programmer´s guide”. United States: The McGraw-Hill Companies Inc, 2008. ISBN: 0-07-159989-4

[18] HASEMAN, Chris. “Android essentials”. United States: Springer-Verlag New York Inc, 2008. ISBN: 978-1-4302-1064-1

[19] Python Programming Language – Official Website. http://www.python.org/ [Consulta año 2011]

[20] PILGRIM, Mark. “Dive into Python”. United States: Springer-Verlag New York Inc, 2004. ISBN: 1-59059-356-1

[21] LAMBERT, Kenneth A.; OSBORNE, Martin. “Fundamentals of Python: first programs”. United States: Cengage Learning, 2011. ISBN: 978-1-111-82270-5

[22] MARUCH, Stef; MARUCH, Aahz. “Python for dummies”. United States: John Wiley & Sons, 2006. ISBN: 978-0-471-77864-6

[23] FERRILL, Paul. “Pro Android Python with SL4A”. United States: Springer-Science+Business Media LLC, 2011. ISBN: 978-1-4302-3569-9

[24] Android-scripting. Scripting Layer for Android brings scripting languages to Android. http://code.google.com/p/android-scripting/ [Consulta año 2011]

[25] JORDAN, Lucas; GREYLING, Pieter. “Practical Android Projects”. United States: Springer-Science+Business Media LLC, 2011. ISBN: 978-1-4302-3243-8

[26] EyeGrade, grading multiple choice questions with a webcam. http://www.it.uc3m.es/jaf/eyegrade/ [Consulta año 2011]

[27] DUDA, Richard O.; HART, Peter E. “Use of the hough transformation to detect lines and curves in pictures”. Communications of the ACM 15 (1) (1972) 11-15.

[28] Eclipse- Official Website. http://www.eclipse.org/ [Consulta año 2011]

Apéndice A

Presupuesto

Se va a proceder a realizar un cálculo aproximado del coste de este proyecto. Comenzando con el desglose de las actividades realizadas e indicando el tiempo empleado en ellas, se llegará finalmente a realizar el presupuesto global del proyecto.

A.1 Descomposición de Actividades

El proyecto que se ha presentado en este documento se ha realizado basándose en las siguientes cuatro fases de actuación:

(1) Fase 1: documentación y estudio del estado del arte para realizar el proyecto.

(2) Fase 2: implementación de la aplicación.

(3) Fase 3: pruebas de la aplicación.

(4) Fase 4: documentación y memoria.

Cada una de las actividades que se han llevado a cabo dentro de dichas fases y el tiempo empleado en ellas, se explican a continuación:

Fase 1: documentación y estudio del estado del arte para realizar el proyecto

(A) Estudio de la plataforma Android y del entorno de desarrollo utilizado, Eclipse. Duración 4 semanas.

(B) Estudio de la herramienta *Scripting Layer for Android* orientado a Python. Duración 4 semanas.

(C) Estudio de la aplicación original EyeGrade implementada en Python. Duración 1 semanas.

(D) Realización de pequeños programas para familiarizarse con la programación del dispositivo. Duración 2 semanas.

Fase 2: implementación de la aplicación.

(A) Implementación de un proyecto Android con SL4A escrito en Python. Duración 4 semanas.

(B) Búsqueda de la biblioteca OpenCV para Python sobre Android. Duración 6 semanas.

(C) Implementación de los menús para la introducción de parámetros de configuración. Duración 3 semanas.

(D) Implementación de la 'creación de exámenes'. Duración 2 semanas.

(E) Implementación de la transición de Python a Java. Duración 2 semanas.

(F) Implementación del procesamiento de imágenes. Duración 4 semanas.

(G) Implementación de la interfaz gráfica de usuario. Duración 2 semanas.

(H) Implementación de la captura de imágenes. Duración 1 semanas.

Fase 3: pruebas de la aplicación.

(A) Pruebas en el emulador del correcto funcionamiento de la primera parte de la aplicación, 'creación de exámenes'. Duración 1 semanas.

(B) Pruebas en el emulador del correcto funcionamiento de la segunda parte de la aplicación usando una imagen de fichero. Duración 2 semanas.

(C) Pruebas en un dispositivo móvil del correcto funcionamiento de la aplicación. Duración 1 semanas.

Fase 4: documentación y memoria.

(A) Redacción de la memoria. Duración 6 semanas.

(B) Corrección y revisión. Duración 2 semanas.

A.2 Resumen y duración

Las actividades y su duración se presentan en forma de resumen en la Tabla A. 1.

Fase		Actividad	Duración(semanas)
Documentación y estudio del estado del arte para realizar el proyecto	1	A	4
		B	4
		C	1
		D	2
Implementación de la aplicación	2	A	4
		B	6
		C	3
		D	2
		E	2
		F	4
		G	2
		H	1
Pruebas de la aplicación	3	A	1
		B	2
		C	1
Documentación y memoria	4	A	6
		B	2
TOTAL			47

Tabla A. 1: Tabla de actividades

A.3 Presupuesto

PRESUPUESTO DEL PROYECTO

Autora: Patricia Uriol Resuela

Departamento: Ingeniería telemática

Descripción del proyecto

- **Título:** MIGRACIÓN A ANDROID DE UNA APLICACIÓN DE VISIÓN ARTIFICIAL PARA EL ÁMBITO EDUCATIVO
- **Duración (meses):** 10,9

Presupuesto total del proyecto (valores en Euros): 40.578,89 euros

Desglose presupuestario (costes directos):

Apellidos y nombre	Categoria	Dedicación (hombres mes)	Coste (hombres mes)	Coste (Euro)
Estévez Ayres, Iria	Ingeniero senior	0,5	4.289,54	2.144,77
Arias Fisteus, Jesús	Ingeniero senior	0,5	4.289,54	2.144,77
Uriol Resuela, Patricia	Ingeniero	10,9	2.694,39	29.368,85
TOTAL				**33.658,39**

Tabla A. 2: Coste directos de personal

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Ordenador portátil	980	100	9	60	147,00
Teléfono móvil Android	210	100	3	60	10,50
TOTAL					**157,50**

Tabla A. 3: Coste equipos

Resumen de costes:

Concepto	Presupuesto Costes Totales
Personal	33.658,39
Amortización	157,50
Costes indirectos (20%)	6.763,00
TOTAL	**40.578,89**

Tabla A. 4: Coste total

El presupuesto total del proyecto asciende a cuarenta mil quinientos setenta y ocho euros con ochenta y nueve céntimos.

Apéndice B

Manual de instalación

El objetivo de este manual es explicar la instalación de la aplicación Eyegrade: visión artificial aplicada al ámbito educativo, en cualquier dispositivo Android. Inicialmente este manual se basa en el uso del emulador o dispositivo virtual del entorno de desarrollo IDE de Eclipse (Ver apartado 5.1) para posteriormente concretar los pasos necesarios para llevar a cabo el uso de la aplicación en un dispositivo físico real.

El desarrollo de aplicaciones Android se puede realizar desde cualquiera de las siguientes plataformas:

- Windows (XP o Vista)[14]
- Mac OS X 10.4.8 o posteriores (solo X86)
- Linux

El SO elegido en el desarrollo de este manual y del proyecto en general, es Windows XP.

1-Descarga e instalación del JDK (Java Development Kit)

Descargar, en función de la plataforma utilizada, la última versión del JDK de Java Sun MicroSystem de la siguiente dirección *web*:

`http://java.sun.com/javase/downloads/index.jsp`

Versión empleada en este proyecto: JDK 6 *Update* 25.

2-Descarga e instalación del SDK

El SDK de Android es *software* libre. Descargar la última versión del SDK en función de la plataforma utilizada, desde la página web de desarrollo Android:

`http://developer.android.com/sdk/index.html`

El SDK se presenta como un archivo ejecutable (versión utilizada r10) o como un archivo comprimido que contiene, además de un ejecutable en su interior, las bibliotecas API, herramientas de desarrollo, documentación y varios ejemplos y demos. Instalar el SDK mediante el archivo ejecutable `.exe` (acción que requiere tiempo, tarda en concluir de instalar todo el paquete SDK) eligiendo conscientemente la carpeta donde se alojará ya que se

[14] Windows 7 ha sido recientemente añadido, y se considera por tanto, otra opción para el desarrollo Android .

necesitará posteriormente (Paso 4.10). En este proyecto, `android-sdk-windows` será la carpeta creada para tal fin.

3-Descarga e instalación de Eclipse

Descarga del *software* libre Eclipse disponible en su propia página principal:

`www.eclipse.org/downloads/`

Versión utilizada Eclipse 3.6.2. La instalación de Eclipse consiste en descomprimir el fichero descargado en una nueva carpeta y ejecutar el fichero `eclipse.exe`. La primera vez que se lleve a cabo esta acción, se creará una carpeta de trabajo, *workspace*, para ubicar los proyectos en desarrollo de Android.

4-Instalación y configuración del *plugin* ADT

Como se vio en el apartado 2.3.3.11, el *plugin* para Eclipse ADT, simplifica el desarrollo de aplicaciones Android mediante la integración de las herramientas de desarrollo. Para realizar su instalación han de seguirse los siguientes pasos:

1) Seleccionar **Help → Install New Software…** desde dentro del Eclipse
2) Dentro del cuadro de dialogo que ha surgido, seleccionar **Add…**
3) En el nuevo cuadro de dialogo (Figura B. 1) añadir la siguiente dirección:
   `https://dl-ssl.google.com/android/eclipse/`

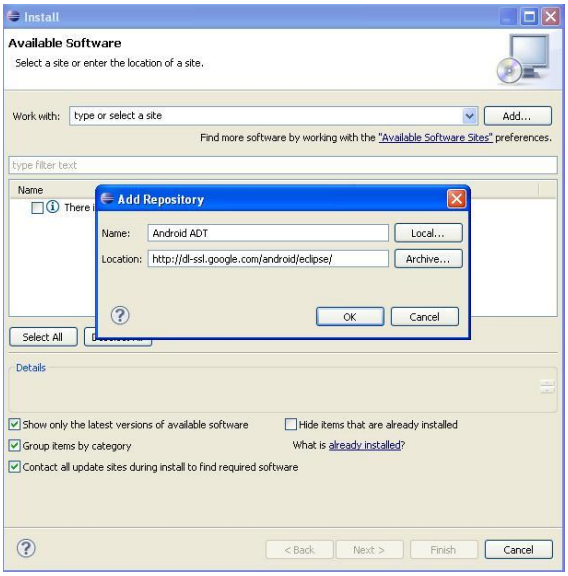


Figura B. 1: Instalación Android ADT

4) Para comprobar la validez de la dirección web, pulsar **Ok**
5) Eclipse procede entonces a descargar el *plugin*. Una vez que haya terminado este proceso, seleccionar los elementos disponibles del resultado de la búsqueda, **Developer Tools**, y pulsar **Next**.
6) Comprobados los detalles de la instalación pulsar **Next**. Leer y **Aceptar** las condiciones del acuerdo de licencia, pulsar **Next** y luego **Finish**.
7) Para hacer efectiva la instalación del *plugin* ADT, cuando se haya concluido la instalación, **reiniciar Eclipse**.

Configuración del *plugin* ADT después de haber realizado su descarga e instalación. Se debe proceder a modificar las preferencias del ADT en el Eclipse indicándole el directorio del SDK de Android:

8) Después del reinicio de Eclipse. Seleccionar **Window → Preferences**
9) En el panel de la izquierda seleccionar **Android**
10) Para añadir la ubicación del SDK (Instalado en el paso 2), pulsar **Browse…** y navegar hasta llegar a la carpeta donde se localizan los archivos del Android SDK. Pulsar entonces **Apply** y **Ok**, Figura B. 2.

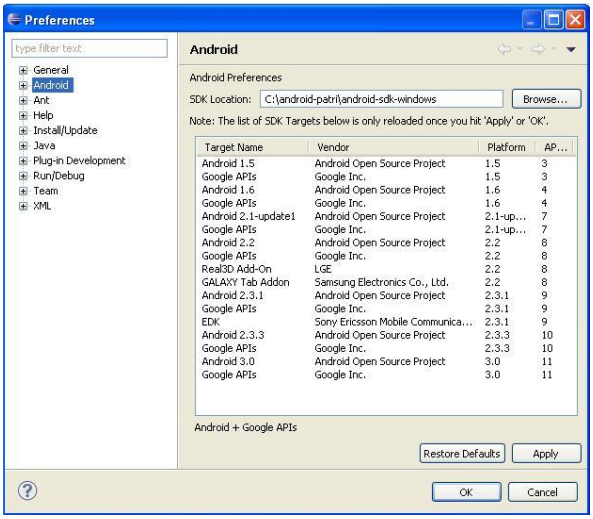


Figura B. 2: Android *preferences*: localización del SDK

En la Figura B. 3 se observan los paquetes instalados. Para acceder a este cuadro de dialogo se selecciona **Window → Android SDK and AVD Manager**. En el panel de la izquierda seleccionar **Installed packages**. En caso de que haya alguna actualización disponible del SDK de Android, la instalación de ésta se puede llevar a cabo en la opción **Available packages** del mismo panel izquierdo.

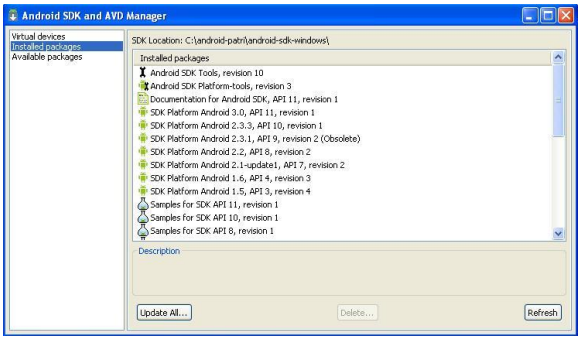


Figura B. 3: Paquetes instalados

5-Creación de un dispositivo virtual Android

Para poder ejecutar las aplicaciones Android se necesita crear un dispositivo virtual AVD según las siguientes instrucciones:

1) Seleccionar **Window → Android SDK and AVD Manager**.
2) En el panel de la izquierda del cuadro de dialogo seleccionar **Virtual devices**, y pulsar el botón **New…**

3) Una vez introducidos los datos del nuevo dispositivo virtual pulsar **Create AVD** (Figura B. 4)

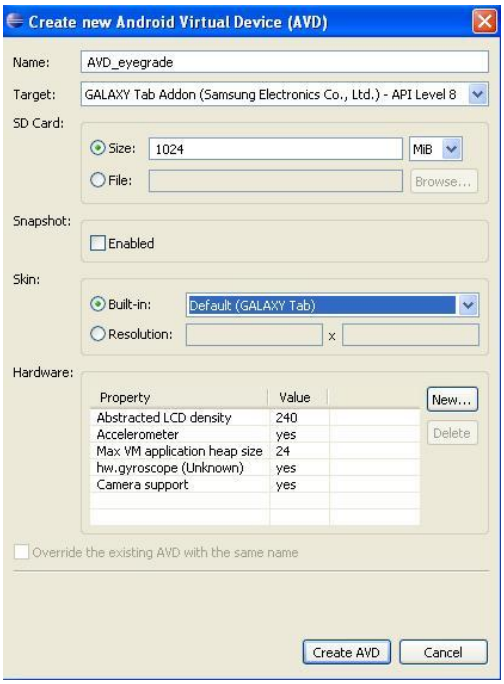


Figura B. 4: Creación del dispositivo virtual AVD

4) Seleccionar el dispositivo virtual y pulsar el botón **Start…**
5) En el nuevo cuadro de dialogo que aparece (Figura B. 5), pulsar en la opción que dice **Scale display to real size** para adaptar el tamaño del dispositivo virtual al tamaño de la pantalla del ordenador con la que se trabaje. Dejar sin marcar la opción **Wipe user data**. Esta opción eliminaría los datos que añada el usuario, y puesto que EyeGrade necesita el uso de otras aplicaciones (paso 6) para su funcionamiento, se debe dejar sin marcar para evitar tener que instalarlas continuamente.
6) Una vez que se tienen configurados todas las opciones pulsar **Launch**.

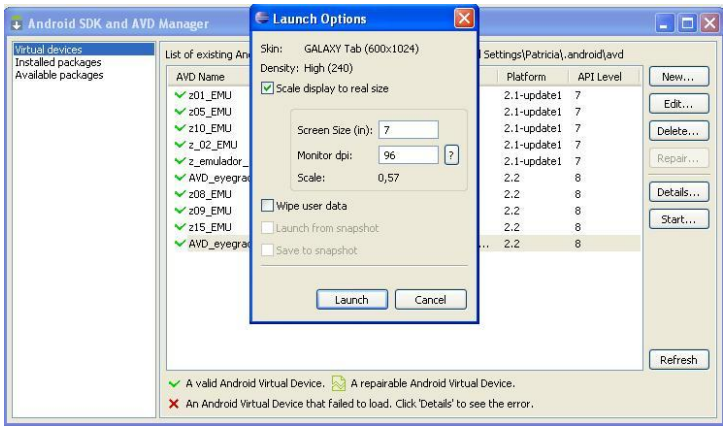
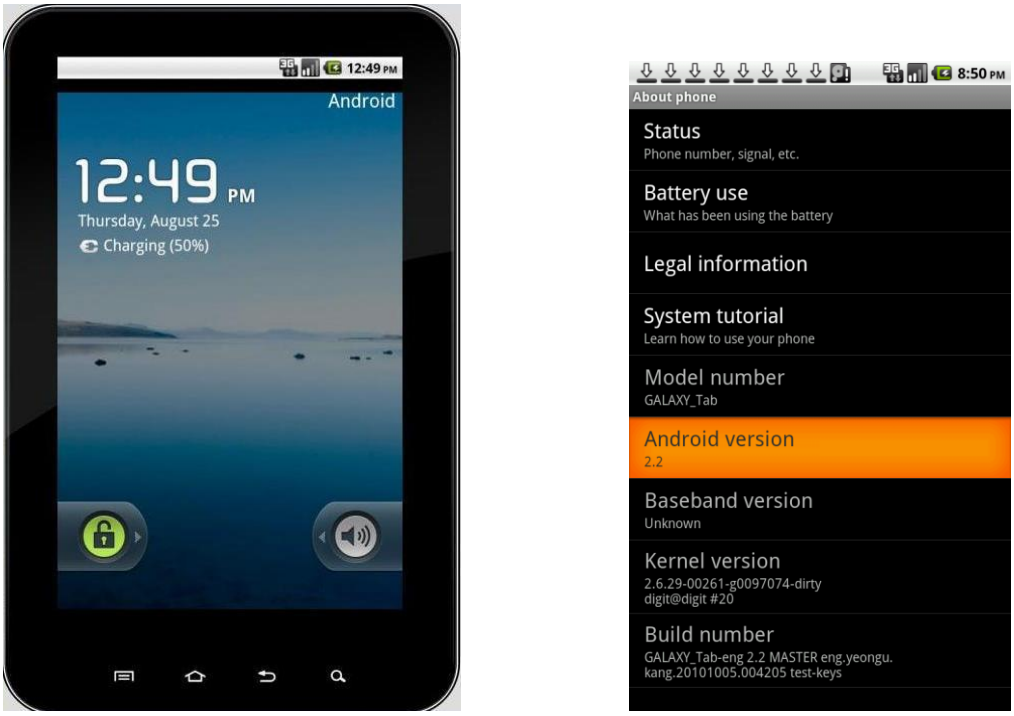


Figura B. 5: AVD Manager, lanzamiento del dispositivo virtual ‘AVD_eyegrade’

En la imagen (a) de la Figura B. 6 se presenta el dispositivo virtual ‘AVD_eyegrade’ usado en el desarrollo de este proyecto. Con una resolución de pantalla de 600x1024 (característica de las pantallas Galaxy Tab de Samsung) y versión de Android 2.2, API *Level* 8 (ver imagen (b) de la Figura B. 6), como principales y más relevantes características para la ejecución de la aplicación EyeGrade.



(a) Pantalla inicial del dispositivo virtual (b) Versión Android 2.2

Figura B. 6: Dispositivo virtual

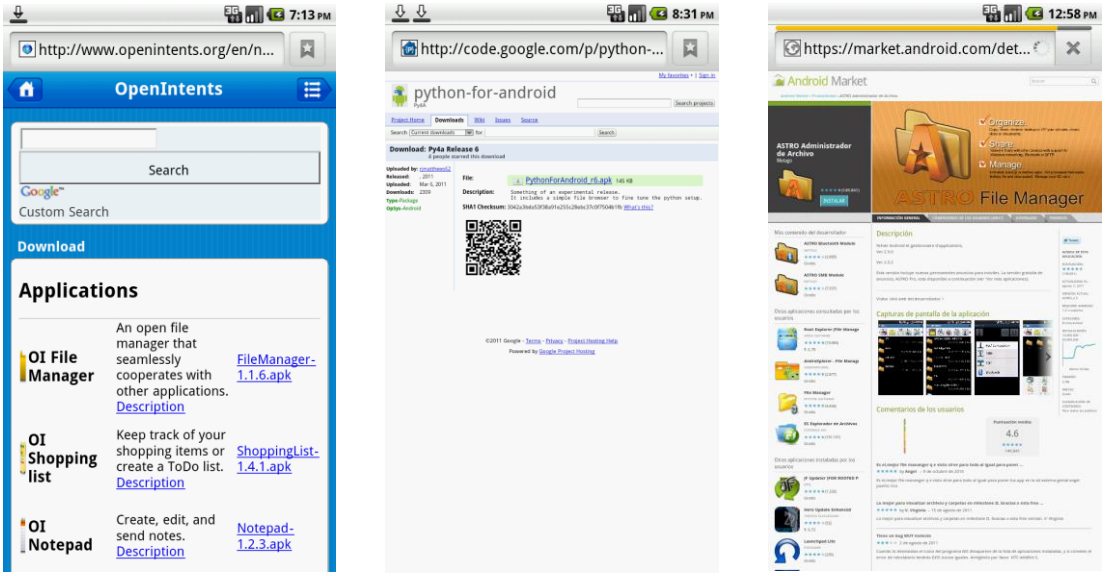
6- Descarga e instalación de aplicaciones externas

La aplicación EyeGrade desarrollada a lo largo de este proyecto, necesita una serie de aplicaciones externas para poder llevar a cabo todas sus funciones correctamente.

- `PythonForAndroid_r6.apk`: el intérprete de Python (Python for Android) utilizado por SL4A para poder interpretar los *script* en lenguaje de programación Python (ver apartado 5.3.1).
- `FileManager-1.1.6.apk`: gestor de directorios (OI File Manager) necesario para la elección de archivos y carpetas en la parte de la UI hecha en Python (ver apartado 5.3.3).
- `ASTRO_File_Manager_2.5.2.apk`: programa necesario para poder descargar y tratar ficheros con extensiones no permitidas usualmente por un dispositivo Android, pero que son necesarias para llevar a cabo la aplicación EyeGrade (ver apartado 5.3.2).

La imagen (a) de la Figura B. 12 muestra el menú de partida del emulador creado en el paso 5. A continuación se va a realizar la descarga e instalación de los programas previamente mencionados. Seguir los siguientes pasos:

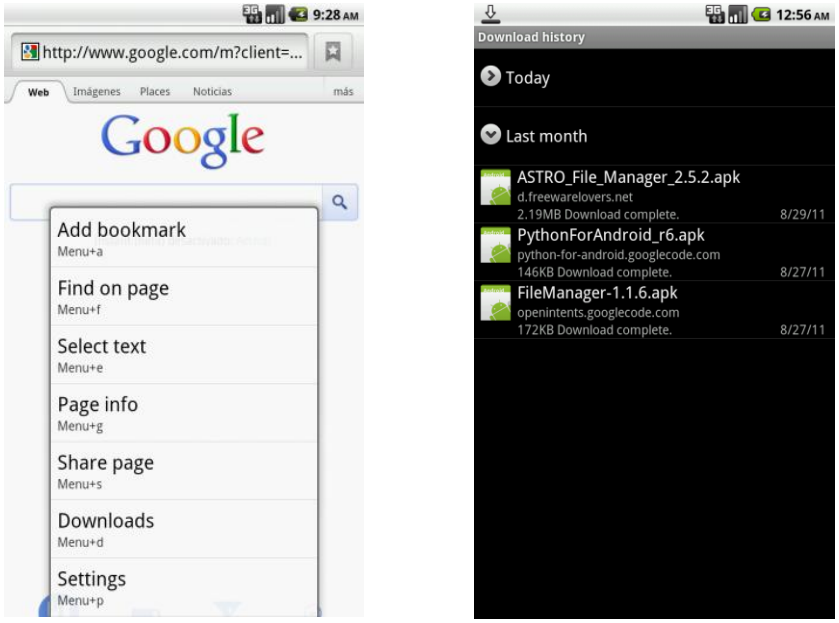
1) Dentro del navegador (**Browser**) de Internet del dispositivo ir a la *web* de OpenIntents y descargar el archivo `FileManager-1.1.6.apk` (Imagen (a) de la Figura B. 7) que se encuentra en la siguiente página:
   http://www.openintents.org/en/download



(a) OI File Manager (b) Python for Android (c) Astro File Manager

Figura B. 7: Descarga de los archivos `.apk` necesarios

2) Continuando en el navegador web, desplazarse a la web de Py4A (*Python for Android*) para descargar (Imagen (b) de la Figura B. 7) el fichero `PythonForAndroid_r6.apk` presente en:
http://code.google.com/p/python-for-android/downloads/detail?name=PythonForAndroid_r6.apk&can=2&q=
3) Dentro de la página del Android Market (Imagen (c) de la Figura B. 7) descargar la tercera aplicación necesaria mediante el archivo `ASTRO_File_Manager_2.5.2.apk` de la página:
https://market.android.com/details?id=com.metago.astro&feature=search_result

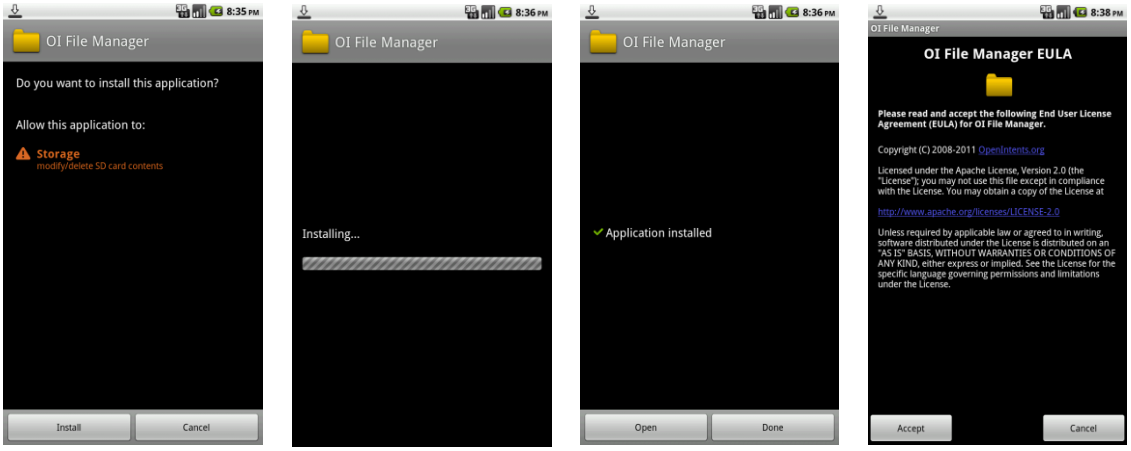


(a) Acceso desde el navegador *web* (b) Historial de descargas

Figura B. 8: Historial de descargas

4) Acceder al historial de descargas (Figura B. 8) y pulsar en el fichero `.apk` del programa OI File Manager: **FileManager-1.1.6.apk**.

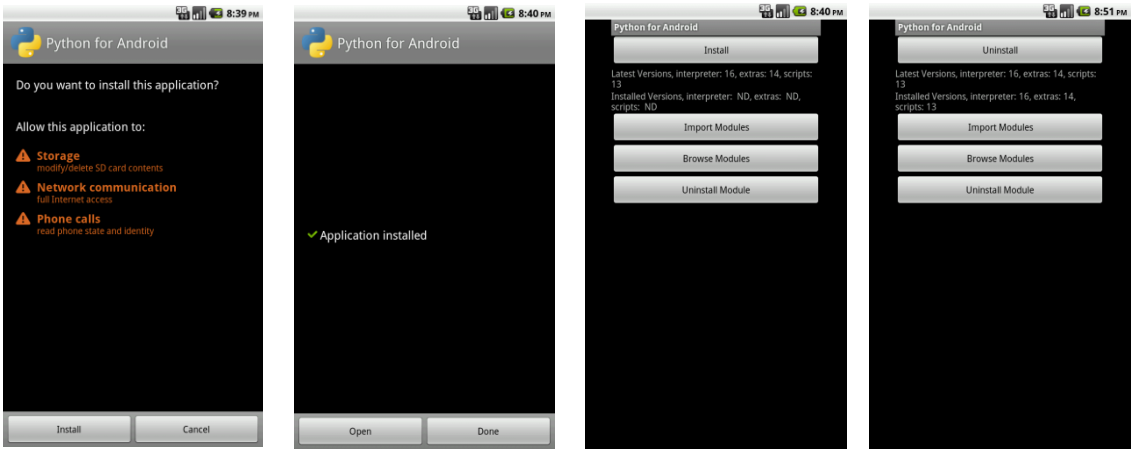
5) En la pantalla que aparece (Imagen (a) de la Figura B. 9), pulsar **Install**.



(a) Inicio de instalación (b) Instalando (c) Aplicación instalada (d) Licencia de uso

Figura B. 9: Proceso de instalación del programa OI File Manager

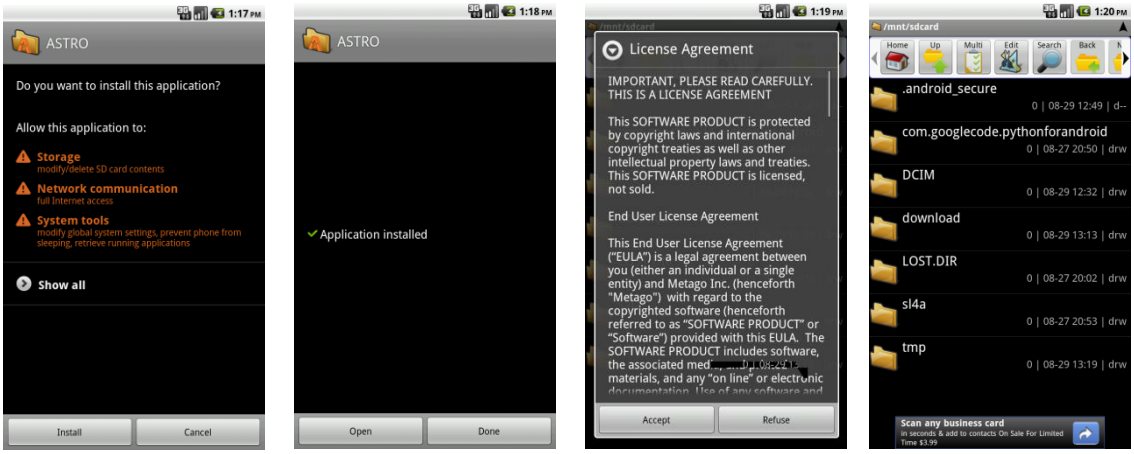
6) Una vez instalada la aplicación OI File Manager (Imagen (c) de la Figura B. 9) pulsar el botón **Open**. Para concluir la instalación correctamente, en la siguiente pantalla (Imagen (d) de la Figura B. 9), se tienen que aceptar las condiciones del acuerdo de licencia mediante el botón **Accept**.
7) Acceder de nuevo al historial de descargas (Figura B. 8) y pulsar en el fichero `.apk` del programa Py4A: **PythonForAndroid_r6.apk**.
8) En la pantalla emergente (Imagen (a) de la Figura B. 10), pulsar **Install**.



(a) Inicio (b) Aplicación instalada (c) Herramientas (d) Py4A instalado

Figura B. 10: Proceso de instalación del programa Python for Android

9) Una vez instalada (Imagen (b) de la Figura B. 10), pulsar el botón **Open**.
10) A continuación hay que descargar y extraer las últimas versiones de las tres herramientas del programa (*interpreter*: `python_r16.zip`, *extras*: `python_extras_r14.zip`, *scripts*: `python_scripts_r13.zip`). Para realizar esta acción se debe pulsar el botón **Install** de la imagen (c) de la Figura B. 10. Una vez que termine este proceso aparecerá la pantalla mostrada en la imagen (d) de la misma figura.
11) Acceder de nuevo al historial de descargas (Figura B. 8) y pulsar en el fichero `.apk` del programa Astro File Manager: **ASTRO_File_Manager_2.5.2.apk**.
12) Pulsar **Install** en la nueva pantalla (Imagen (a) de la Figura B. 11Figura B. **10**).



(a) Inicio (b) Aplicación instalada (c) Licencia de uso (d) Aplicación instalada

Figura B. 11: Proceso de instalación del programa Astro File Manager

13) Una vez instalada (Imagen (b) de la Figura B. 11), pulsar el botón **Open.**
14) En la imagen (c) de la Figura B. 11 leer y aceptar las condiciones de uso pulsando **Accept**.

Una vez concluida la instrucción anterior, ya se ha completado la instalación de las aplicaciones necesarias para poder ejecutar EyeGrade. La imagen (b) de la Figura B. 12 muestra cómo debe quedar el menú después de realizar la instalación de estas tres aplicaciones, ASTRO (parte superior), OI File Manager y Python for Android (segunda fila inferior).

(a) Al crearse el AVD (b) Instaladas aplicaciones complementarias (c) Instalado EyeGrade

Figura B. 12: Evolución del menú Android del emulador

7- Creación de un '*Launch Configuration*'

'*Launch Configuration*' permite especificar ciertas opciones a la hora de ejecutar o depurar aplicaciones con Eclipse. Opciones como por ejemplo la aplicación que se va a lanzar o el emulador a utilizar. Con los siguientes pasos se ha de crear la configuración para la aplicación EyeGrade:

1) Seleccionar **Run → Run Configurations…** dentro de Eclipse.

2) En la nueva pantalla, seleccionar de la lista de tipos de proyectos **Android Application**, y con el botón derecho seleccionar **New**.
3) Introducir el nombre para la configuración (título descriptivo del *setup*) en **Name**. Elegido en este caso el mismo que la aplicación, EyeGrade.

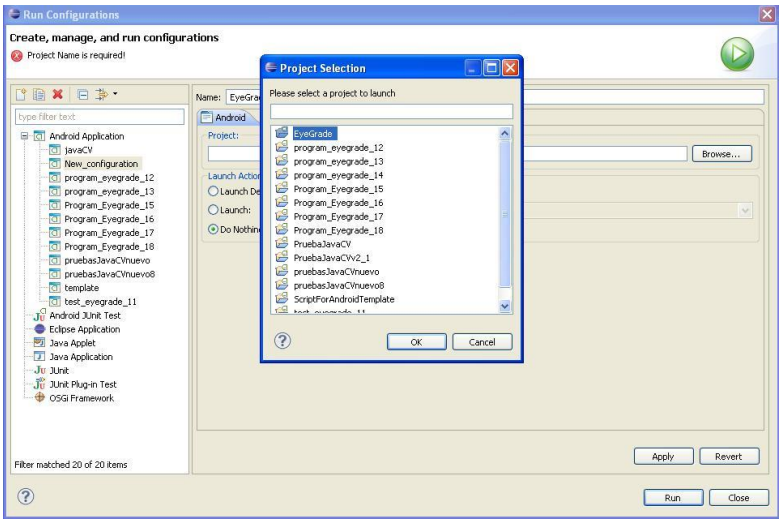


Figura B. 13: Creación de un nuevo '*Launch Configuration*' para la aplicación EyeGrade

4) En la primera pestaña, **Android**, (Figura B. 13) se elige el proyecto y la *activity* que comenzará cuando se lance la aplicación. Seleccionar el proyecto pulsando el botón **Browse… → EyeGrade.** Seleccionar la opción **Do Nothing** para que no comience la aplicación hasta que no se pulse en su icono del emulador.
5) En la pestaña **Target** se elige el emulador que se va a utilizar. Seleccionar **Manual**, para poder elegir el dispositivo virtual cada vez que se vaya a ejecutar o depurar la aplicación. En la pestaña **Common** se seleccionan **Debug** y **Run** para mostrar la configuración en ambos menús.
6) Seleccionar **Apply → Close**.

8- Instalación de la aplicación en el dispositivo virtual

1) Seleccionar **EyeGrade** en el menú desplegable del icono **Run**, Figura B. 14.

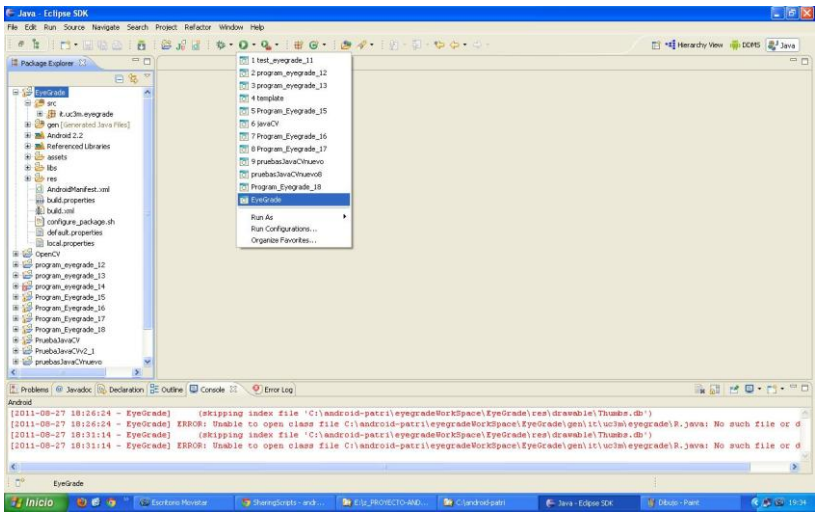


Figura B. 14: Selección de aplicación en el menú desplegable *Run*

2) Seleccionar el dispositivo virtual creado y lanzado en el paso 5, Figura B. 15.

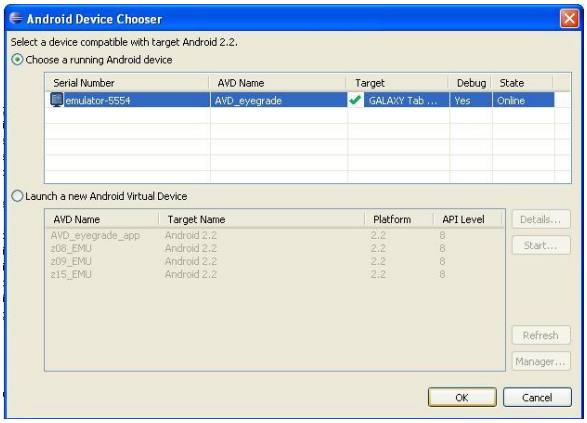


Figura B. 15: Selección del dispositivo virtual Android

Una vez concluida la instalación de la aplicación en el dispositivo virtual, la apariencia de éste será como la imagen (c) de la Figura B. 12.

9- Instalación y ejecución de la aplicación en el dispositivo físico Android

1) Seleccionar **EyeGrade** en el menú desplegable del icono **Run**, Figura B. 14.
2) Seleccionar el dispositivo físico Android conectado al ordenador, Figura B. 16.

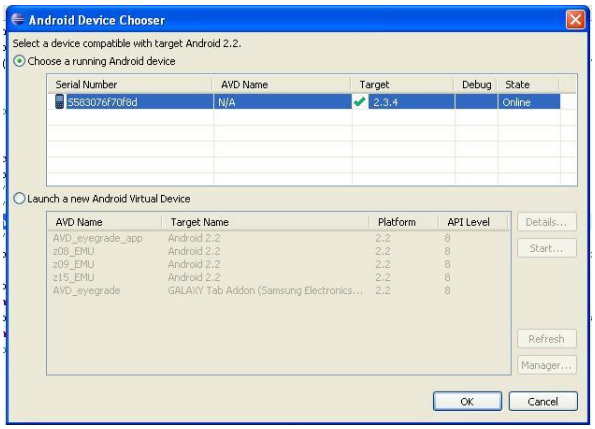


Figura B. 16: Selección del dispositivo físico de Android

3) Seleccionar el icono de la aplicación EyeGrade, Figura B. 17.

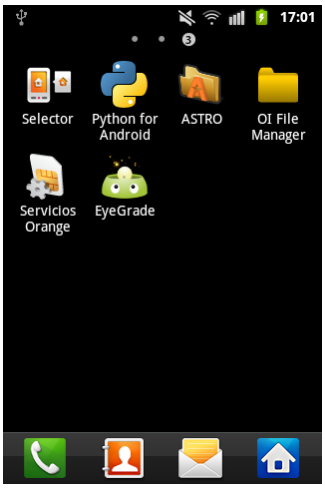


Figura B. 17: Aplicación EyeGrade instalada en el dispositivo físico